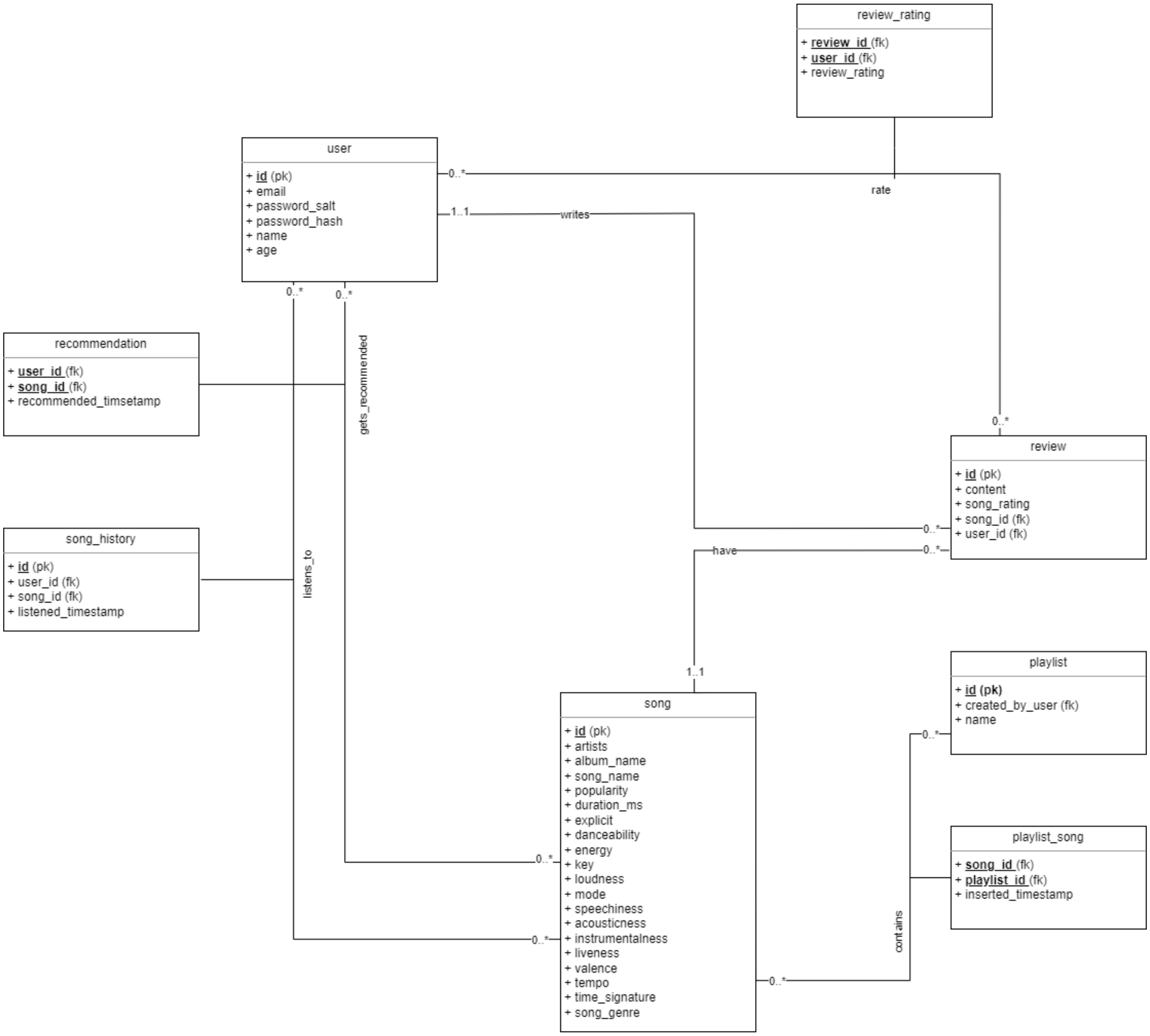
Changes made according to the feedback received for the previous stage (Stage 2):

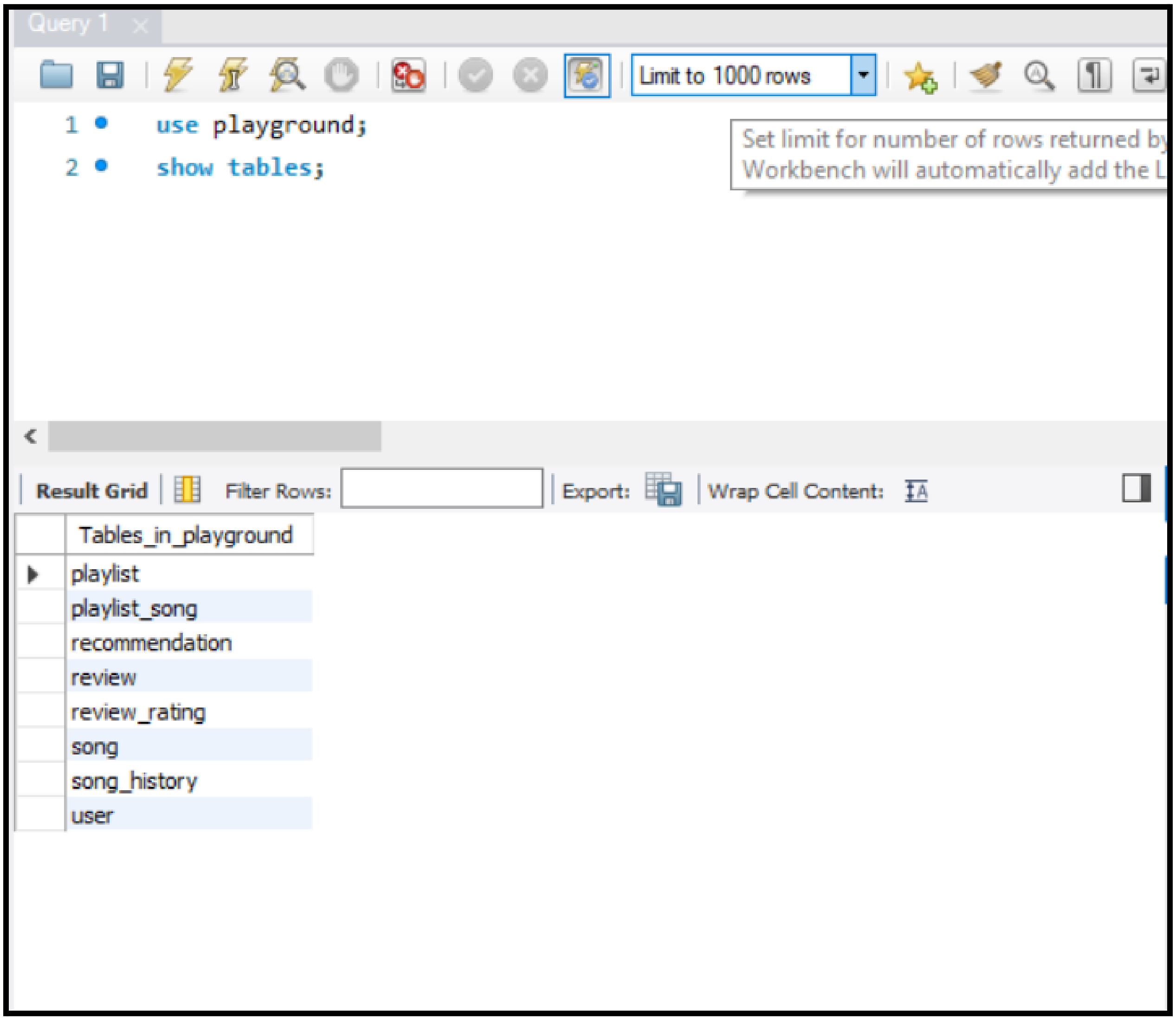


-The review_ratings table has been converted to a relationship.
- All unnecessary primary keys were removed from relationship tables.
- Song history table needs id to be its primary key since the user can listen to the same song multiple times, so the combination of user_id and song_id cannot uniquely identify tuples.

STAGE 3:

1. **Implementation of tables include:**
    a. Song
    b. User
    c. Playlist
    d. Review
    e. song_history
    f. recommendation
    g. playlist_song
    h. review_rating

Tables in Database


Query 1
Limit to 1000 rows
Set limit for number of rows returned by
Workbench will automatically add the L
Result Grid
Filter Rows:
Export:
Wrap Cell Content:

```
use playground;
show tables;
```

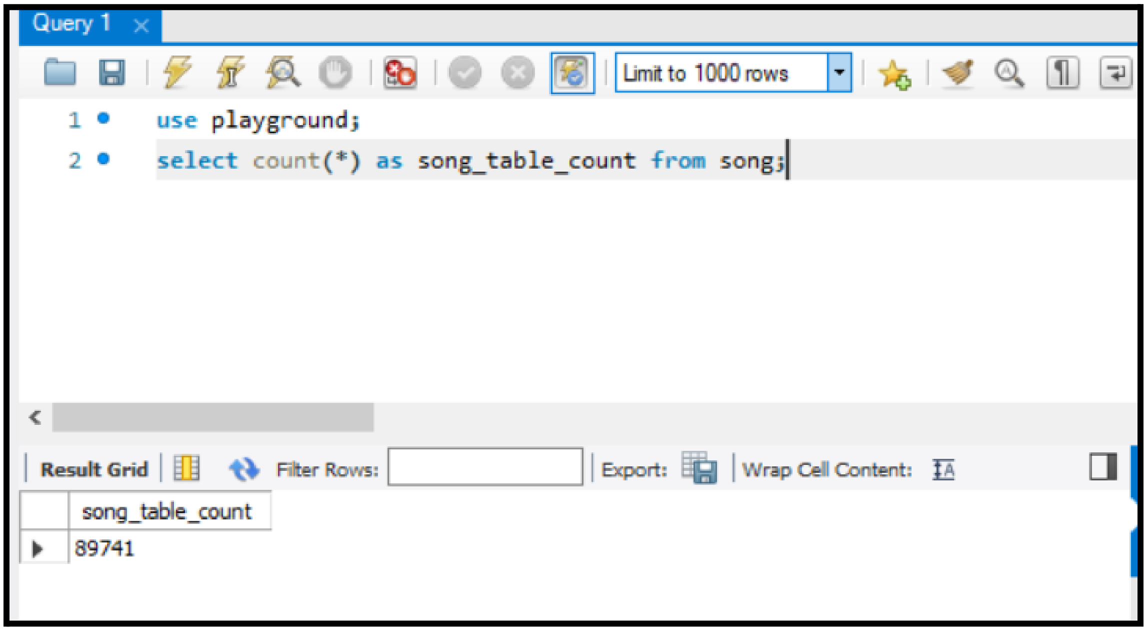
Tables_in_playground
playlist
playlist_song
recommendation
review
review_rating
song
song_history
user

2. DDL commands for table:

a. Table- Song:

```
CREATE TABLE song
( id VARCHAR(22),
artist VARCHAR(100),
album_name VARCHAR(100),
song_name VARCHAR(100),
popularity INT,
duration_ms INT,
explicit boolean,
danceability DECIMAL(20,10),
energy DECIMAL(20,10),
`key` INT,
loudness DECIMAL(20,10),
mode INT,
speechiness DECIMAL(20,10),
acousticness DECIMAL(20,10),
instrumentalness DECIMAL(20,10),
liveness DECIMAL(20,10),
valence DECIMAL(20,10),
tempo DECIMAL(20,10),
time_signature INT,
song_genre VARCHAR(50),
PRIMARY KEY (id));
```

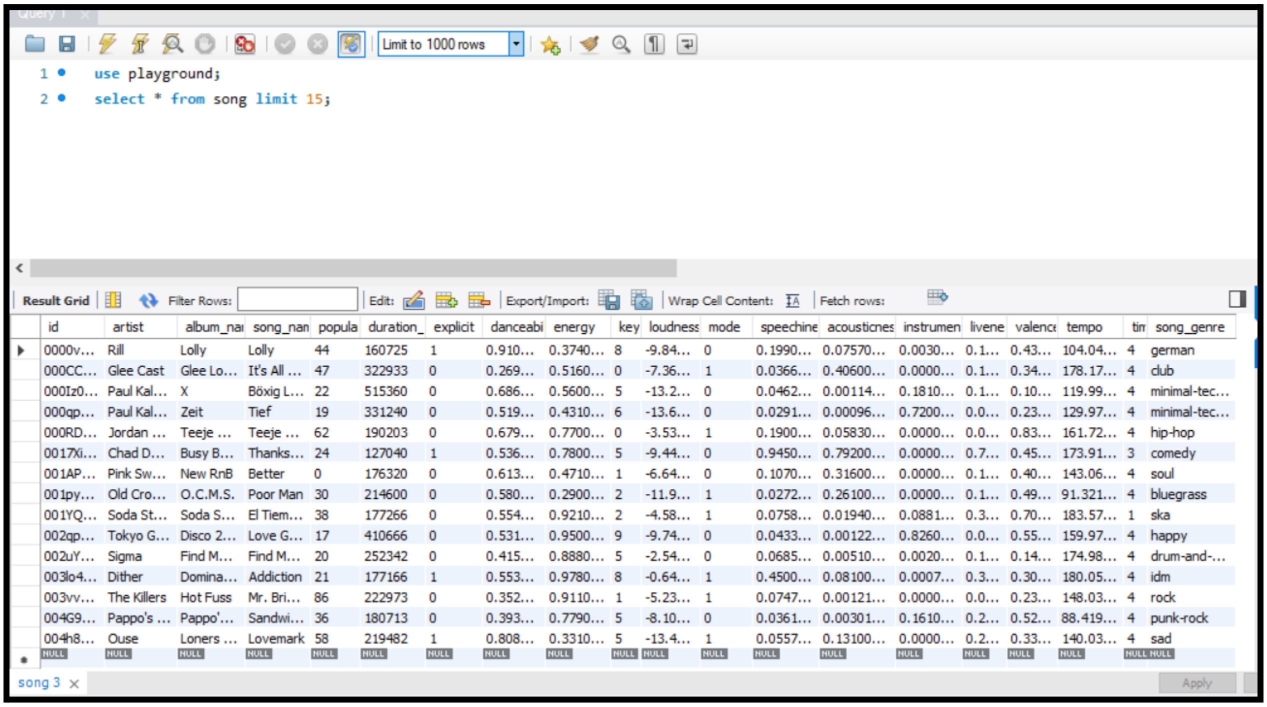
Count of rows for this table:



```
use playground;
select count(*) as song_table_count from song;
```

song_table_count
89741

Top 15 rows in the table:



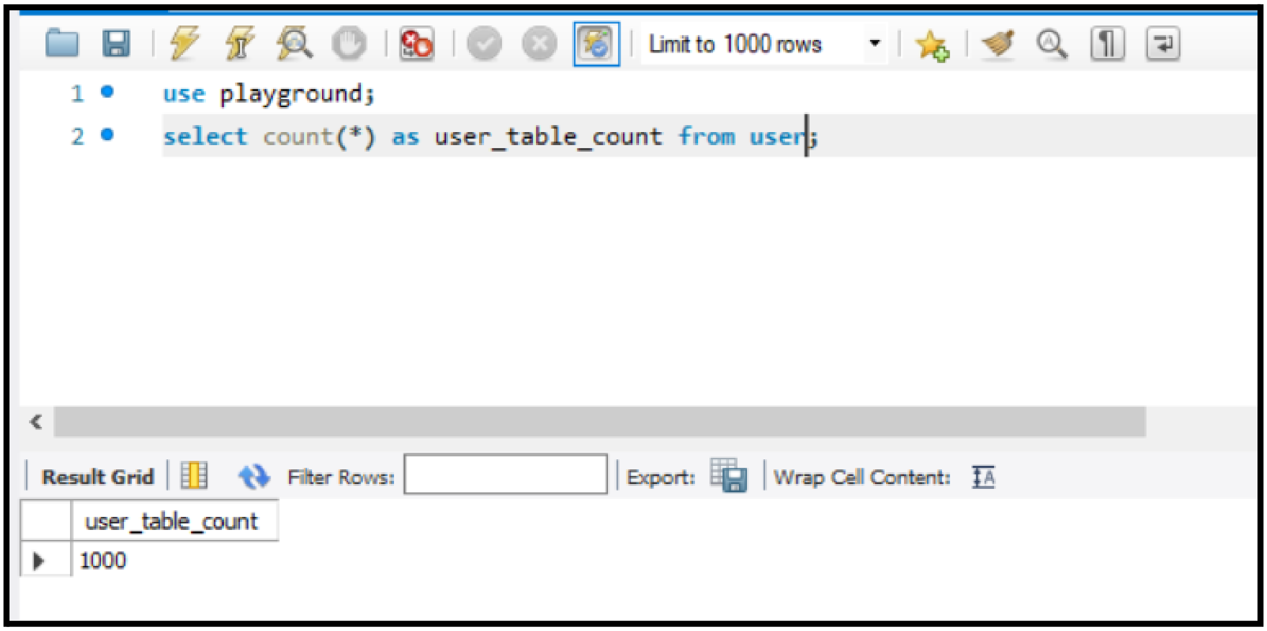
```
use playground;
select * from song limit 15;
```

id	artist	album_nai	song_nan	popula	duration_	explicit	danceabi	energy	key	loudness	mode	speechine	acousticnes	instrumen	livene	valence	tempo	tim	song_genre
0000v...	Rill	Lolly	Lolly	44	160725	1	0.910...	0.3740...	8	-9.84...	0	0.1990...	0.07570...	0.0030...	0.1...	0.43...	104.04...	4	german
000CC...	Glee Cast	Glee Lo...	It's All ...	47	322933	0	0.269...	0.5160...	0	-7.36...	1	0.0366...	0.40600...	0.0000...	0.1...	0.34...	178.17...	4	club
000Iz0...	Paul Kal...	X	Böxig L...	22	515360	0	0.686...	0.5600...	5	-13.2...	0	0.0462...	0.00114...	0.1810...	0.1...	0.10...	119.99...	4	minimal-tec...
000qp...	Paul Kal...	Zeit	Tief	19	331240	0	0.519...	0.4310...	6	-13.6...	0	0.0291...	0.00096...	0.7200...	0.0...	0.23...	129.97...	4	minimal-tec...
000RD...	Jordan ...	Teeje ...	Teeje ...	62	190203	0	0.679...	0.7700...	0	-3.53...	1	0.1900...	0.05830...	0.0000...	0.0...	0.83...	161.72...	4	hip-hop
0017Xi...	Chad D...	Busy B...	Thanks...	24	127040	1	0.536...	0.7800...	5	-9.44...	0	0.9450...	0.79200...	0.0000...	0.7...	0.45...	173.91...	3	comedy
001AP...	Pink Sw...	New RnB	Better	0	176320	0	0.613...	0.4710...	1	-6.64...	0	0.1070...	0.31600...	0.0000...	0.1...	0.40...	143.06...	4	soul
001py...	Old Cro...	O.C.M.S.	Poor Man	30	214600	0	0.580...	0.2900...	2	-11.9...	1	0.0272...	0.26100...	0.0000...	0.1...	0.49...	91.321...	4	bluegrass
001YQ...	Soda St...	Soda S...	El Tiem...	38	177266	0	0.554...	0.9210...	2	-4.58...	1	0.0758...	0.01940...	0.0881...	0.3...	0.70...	183.57...	1	ska
002qp...	Tokyo G...	Disco 2...	Love G...	17	410666	0	0.531...	0.9500...	9	-9.74...	0	0.0433...	0.00122...	0.8260...	0.0...	0.55...	159.97...	4	happy
002uY...	Sigma	Find M...	Find M...	20	252342	0	0.415...	0.8880...	5	-2.54...	0	0.0685...	0.00510...	0.0020...	0.1...	0.14...	174.98...	4	drum-and-...
003lo4...	Dither	Domina...	Addiction	21	177166	1	0.553...	0.9780...	8	-0.64...	1	0.4500...	0.08100...	0.0007...	0.3...	0.30...	180.05...	4	idm
003vv...	The Killers	Hot Fuss	Mr. Bri...	86	222973	0	0.352...	0.9110...	1	-5.23...	1	0.0747...	0.00121...	0.0000...	0.0...	0.23...	148.03...	4	rock
004G9...	Pappo's ...	Pappo'...	Sandwi...	36	180713	0	0.393...	0.7790...	5	-8.10...	0	0.0361...	0.00301...	0.1610...	0.2...	0.52...	88.419...	4	punk-rock
004h8...	Ouse	Loners ...	Lovemark	58	219482	1	0.808...	0.3310...	5	-13.4...	1	0.0557...	0.13100...	0.0000...	0.2...	0.33...	140.03...	4	sad
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

b. Table- User:

```
CREATE TABLE user(
id VARCHAR(100),
email VARCHAR(150),
name VARCHAR(50),
age INT,
password_salt VARCHAR(32),
password_hash VARCHAR(32),
PRIMARY KEY (id)
);
```

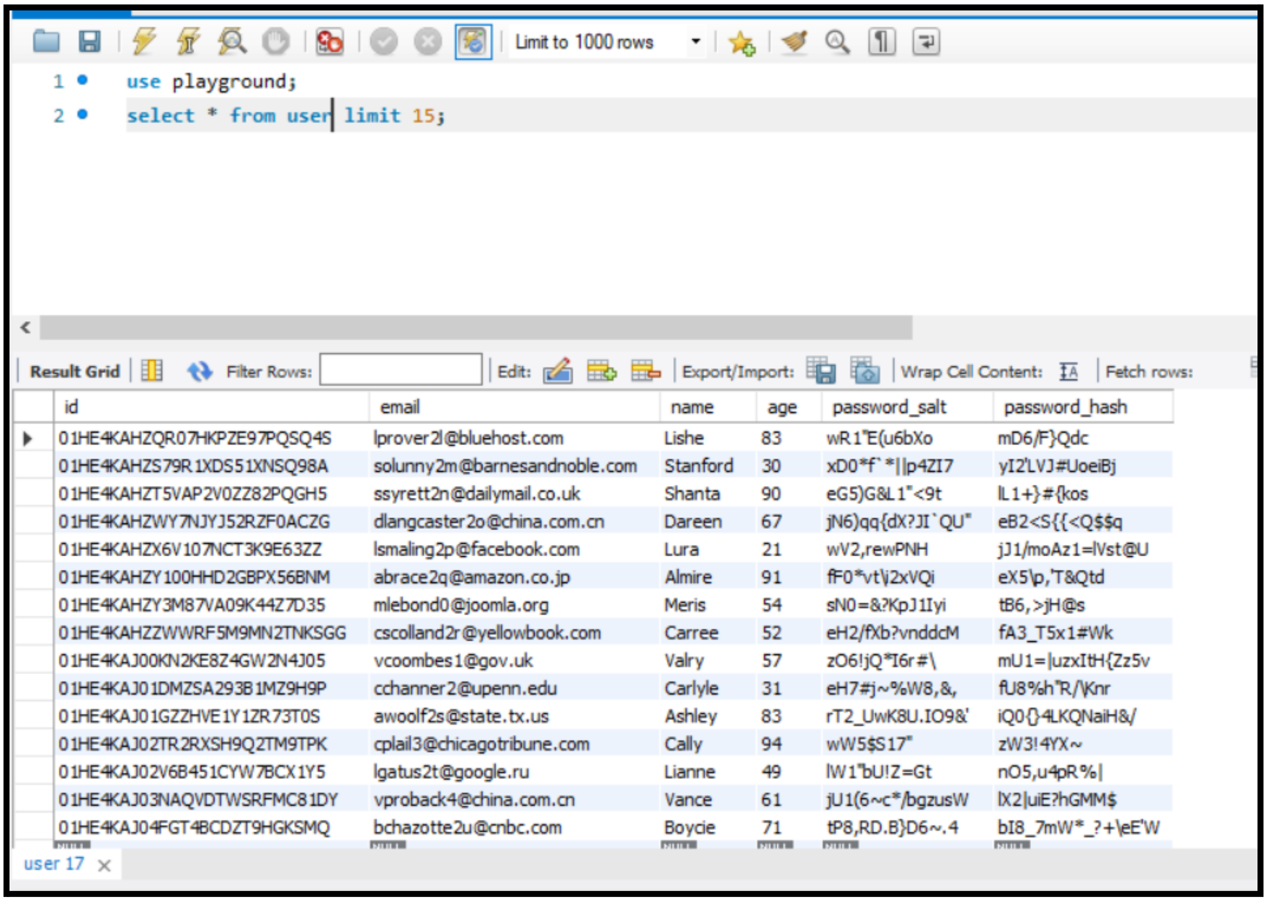
Count of rows for this table:



```
use playground;
select count(*) as user_table_count from user;
```

	user_table_count
▶	1000

Top 15 rows in the table:



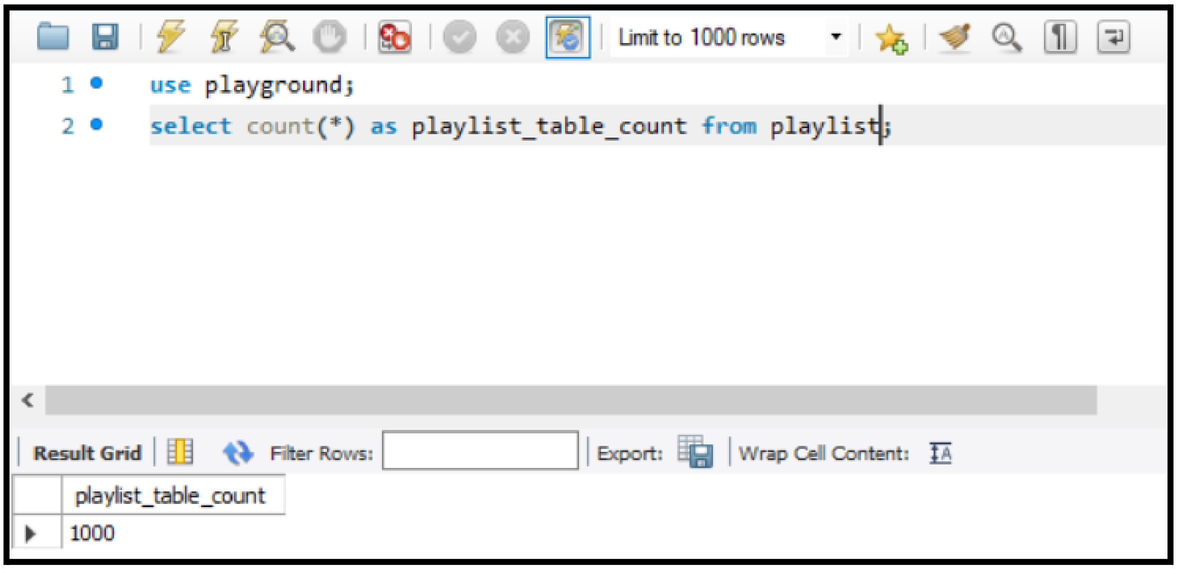
```
use playground;
select * from user limit 15;
```

	id	email	name	age	password_salt	password_hash
▶	01HE4KAHZQR07HKPZE97PQSQ4S	lprover2l@bluehost.com	Lishe	83	wR1"E(u6bXo	mD6/F}Qdc
	01HE4KAHZS79R1XDS51XNSQ98A	solunny2m@barnesandnoble.com	Stanford	30	xD0*f`*||p4ZI7	yI2'LVJ#UoeiBj
	01HE4KAHZT5VAP2V0ZZ82PQGH5	ssyrett2n@dailymail.co.uk	Shanta	90	eG5)G&L1"<9t	lL1+}#{kos
	01HE4KAHZWY7NJYJ52RZF0ACZG	dlangcaster2o@china.com.cn	Dareen	67	jN6)qq{dX?JI`QU"	eB2<S{{<Q$$q
	01HE4KAHZX6V107NCT3K9E63ZZ	lsmaling2p@facebook.com	Lura	21	wV2,rewPNH	jJ1/moAz1=lVst@U
	01HE4KAHZY100HHD2GBPX56BNM	abrace2q@amazon.co.jp	Almire	91	fF0*vt\j2xVQi	eX5\p,'T&Qtd
	01HE4KAHZY3M87VA09K44Z7D35	mlebond0@joomla.org	Meris	54	sN0=&?KpJ1Iyi	tB6,>jH@s
	01HE4KAHZZWWRF5M9MN2TNKSGG	cscolland2r@yellowbook.com	Carree	52	eH2/fXb?vnddcM	fA3_T5x1#Wk
	01HE4KAJ00KN2KE8Z4GW2N4J05	vcoombes1@gov.uk	Valry	57	zO6!jQ*I6r#\	mU1=|uzxItH{Zz5v
	01HE4KAJ01DMZSA293B1MZ9H9P	cchanner2@upenn.edu	Carlyle	31	eH7#j~%W8,&,	fU8%h"R/\Knr
	01HE4KAJ01GZZHVE1Y1ZR73T0S	awoolf2s@state.tx.us	Ashley	83	rT2_UwK8U.IO9&'	iQ0{}4LKQNaiH&/
	01HE4KAJ02TR2RXSH9Q2TM9TPK	cplail3@chicagotribune.com	Cally	94	wW5$S17"	zW3!4YX~
	01HE4KAJ02V6B451CYW7BCX1Y5	lgatus2t@google.ru	Lianne	49	lW1"bU!Z=Gt	nO5,u4pR%|
	01HE4KAJ03NAQVDTWSRFMC81DY	vproback4@china.com.cn	Vance	61	jU1(6~c*/bgzusW	lX2|uiE?hGMM$
	01HE4KAJ04FGT4BCDZT9HGKSMQ	bchazotte2u@cnbc.com	Boycie	71	tP8,RD.B}D6~.4	bI8_7mW*_?+\eE'W

c. Table- Playlist:

```
CREATE TABLE playlist(
id VARCHAR(100),
created_by_user VARCHAR(100),
name VARCHAR(100),
PRIMARY KEY (id),
FOREIGN KEY (created_by_user ) REFERENCES user(id) ON DELETE CASCADE);
```

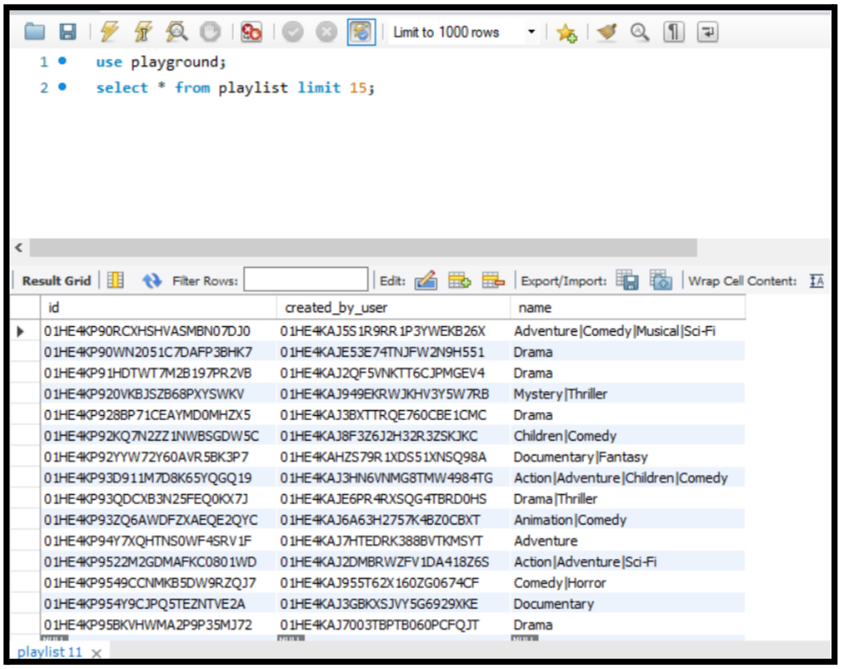
Count of rows for this table:



```
use playground;
select count(*) as playlist_table_count from playlist;
```

playlist_table_count
1000

Top 15 rows in the table:



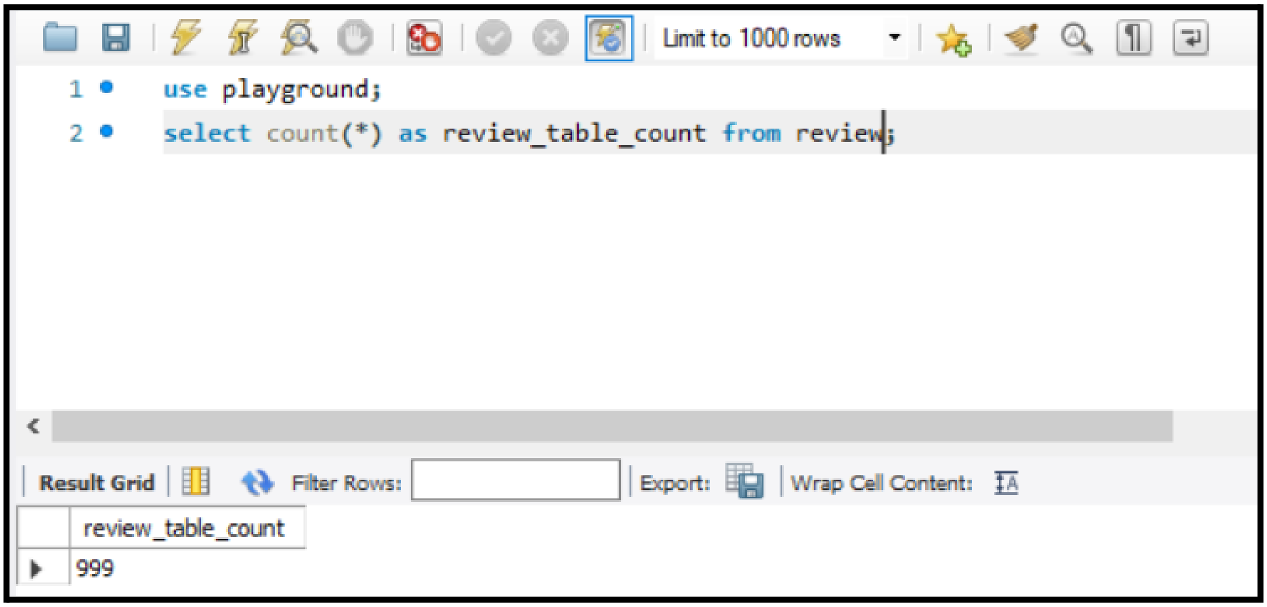
```
use playground;
select * from playlist limit 15;
```

id	created_by_user	name
01HE4KP90RCXHSHVASMBN07DJ0	01HE4KAJ5S1R9RR1P3YWEKB26X	Adventure|Comedy|Musical|Sci-Fi
01HE4KP90WN2051C7DAFP3BHK7	01HE4KAJE53E74TNJFW2N9H551	Drama
01HE4KP91HDTWT7M2B197PR2VB	01HE4KAJ2QF5VNKTT6CJPMGEV4	Drama
01HE4KP920VKBJSZB68PXYSWKV	01HE4KAJ949EKRWJKHV3Y5W7RB	Mystery|Thriller
01HE4KP928BP71CEAYMD0MHZX5	01HE4KAJ3BXTTRQE760CBE1CMC	Drama
01HE4KP92KQ7N2ZZ1NWBSGDW5C	01HE4KAJ8F3Z6J2H32R3ZSKJKC	Children|Comedy
01HE4KP92YYW72Y60AVR5BK3P7	01HE4KAHZS79R1XDS51XNSQ98A	Documentary|Fantasy
01HE4KP93D911M7D8K65YQGQ19	01HE4KAJ3HN6VNMG8TMW4984TG	Action|Adventure|Children|Comedy
01HE4KP93QDCXB3N25FEQ0KX7J	01HE4KAJE6PR4RXSQG4TBRD0HS	Drama|Thriller
01HE4KP93ZQ6AWDFZXAEQE2QYC	01HE4KAJ6A63H2757K4BZ0CBXT	Animation|Comedy
01HE4KP94Y7XQHTNS0WF4SRV1F	01HE4KAJ7HTEDRK388BVTKMSYT	Adventure
01HE4KP9522M2GDMAFKC0801WD	01HE4KAJ2DMBRWZFV1DA418Z6S	Action|Adventure|Sci-Fi
01HE4KP9549CCNMKB5DW9RZQJ7	01HE4KAJ955T62X160ZG0674CF	Comedy|Horror
01HE4KP954Y9CJPQ5TEZNTVE2A	01HE4KAJ3GBKXSJVY5G6929XKE	Documentary
01HE4KP95BKVHWMA2P9P35MJ72	01HE4KAJ7003TBPTB060PCFQJT	Drama

d. Table- Review:

```
CREATE TABLE review(
id VARCHAR(100),
content VARCHAR(1000),
song_rating INT,
song_id VARCHAR(22),
user_id VARCHAR(100),
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE,
FOREIGN KEY (song_id) REFERENCES song(id) ON DELETE CASCADE);
```

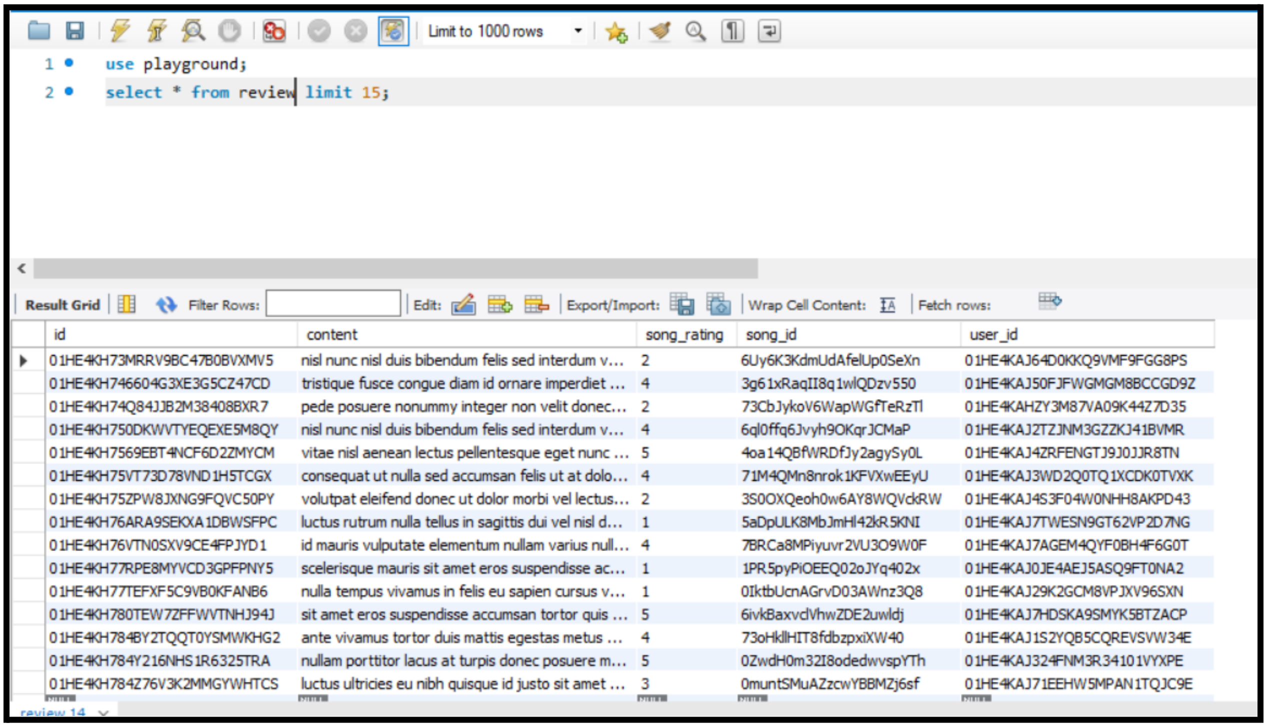
Count of rows for this table:



```
use playground;
select count(*) as review_table_count from review;
```

```
review_table_count
999
```

Top 15 rows in the table:



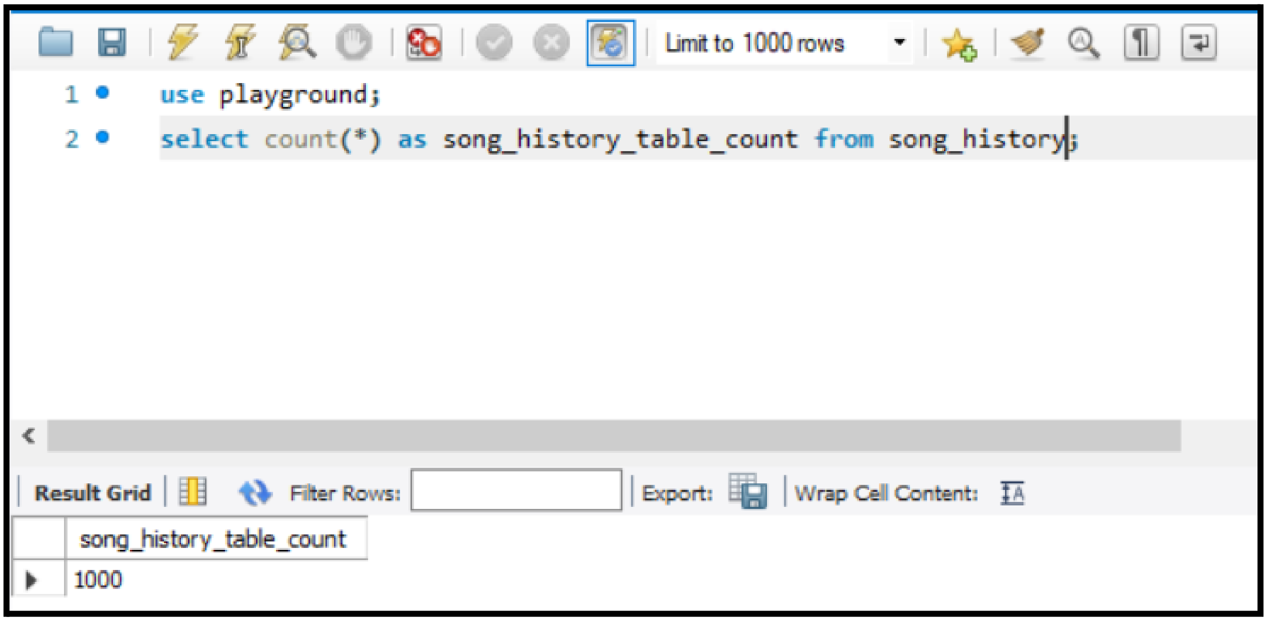
```
use playground;
select * from review limit 15;
```

id	content	song_rating	song_id	user_id
01HE4KH73MRRV9BC47B0BVXMV5	nisl nunc nisl duis bibendum felis sed interdum v...	2	6Uy6K3KdmUdAfelUp0SeXn	01HE4KAJ64D0KKQ9VMF9FGG8PS
01HE4KH746604G3XE3G5CZ47CD	tristique fusce congue diam id ornare imperdiet ...	4	3g61xRaqII8q1wlQDzv550	01HE4KAJ50FJFWGMGM8BCCGD9Z
01HE4KH74Q84JJB2M38408BXR7	pede posuere nonummy integer non velit donec...	2	73CbJykoV6WapWGfTeRzTl	01HE4KAHZY3M87VA09K44Z7D35
01HE4KH750DKWVTYEQEXE5M8QY	nisl nunc nisl duis bibendum felis sed interdum v...	4	6ql0ffq6Jvyh9OKqrJCMaP	01HE4KAJ2TZJNM3GZZKJ41BVMR
01HE4KH7569EBT4NCF6D2ZMYCM	vitae nisl aenean lectus pellentesque eget nunc ...	5	4oa14QBfWRDfJy2agySy0L	01HE4KAJ4ZRFENGTJ9J0JJR8TN
01HE4KH75VT73D78VND1H5TCGX	consequat ut nulla sed accumsan felis ut at dolo...	4	71M4QMn8nrok1KFVXwEEyU	01HE4KAJ3WD2Q0TQ1XCDK0TVXK
01HE4KH75ZPW8JXNG9FQVC50PY	volutpat eleifend donec ut dolor morbi vel lectus...	2	3S0OXQeoh0w6AY8WQVckRW	01HE4KAJ4S3F04W0NHH8AKPD43
01HE4KH76ARA9SEKXA1DBWSFPC	luctus rutrum nulla tellus in sagittis dui vel nisl d...	1	5aDpULK8MbJmHl42kR5KNI	01HE4KAJ7TWESN9GT62VP2D7NG
01HE4KH76VTN0SXV9CE4FPJYD1	id mauris vulputate elementum nullam varius null...	4	7BRCa8MPiyuvr2VU3O9W0F	01HE4KAJ7AGEM4QYF0BH4F6G0T
01HE4KH77RPE8MYVCD3GPFPNY5	scelerisque mauris sit amet eros suspendisse ac...	1	1PR5pyPiOEEQ02oJYq402x	01HE4KAJ0JE4AEJ5ASQ9FT0NA2
01HE4KH77TEFXF5C9VB0KFANB6	nulla tempus vivamus in felis eu sapien cursus v...	1	0IktbUcnAGrvD03AWnz3Q8	01HE4KAJ29K2GCM8VPJXV96SXN
01HE4KH780TEW7ZFFWVTNHJ94J	sit amet eros suspendisse accumsan tortor quis ...	5	6ivkBaxvdVhwZDE2uwldj	01HE4KAJ7HDSKA9SMYK5BTZACP
01HE4KH784BY2TQQT0YSMWKHG2	ante vivamus tortor duis mattis egestas metus ...	4	73oHkllHIT8fdbzpxiXW40	01HE4KAJ1S2YQB5CQREVSVW34E
01HE4KH784Y216NHS1R6325TRA	nullam porttitor lacus at turpis donec posuere m...	5	0ZwdH0m32I8odedwvspYTh	01HE4KAJ324FNM3R34101VYXPE
01HE4KH784Z76V3K2MMGYWHTCS	luctus ultricies eu nibh quisque id justo sit amet ...	3	0muntSMuAZzcwYBBMZj6sf	01HE4KAJ71EEHW5MPAN1TQJC9E

e. Table- song_history:

```
CREATE TABLE song_history(
id VARCHAR(100),
user_id VARCHAR(100),
song_id VARCHAR(22),
listened_timestamp TIMESTAMP,
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE,
FOREIGN KEY (song_id) REFERENCES song(id) ON DELETE CASCADE);
```

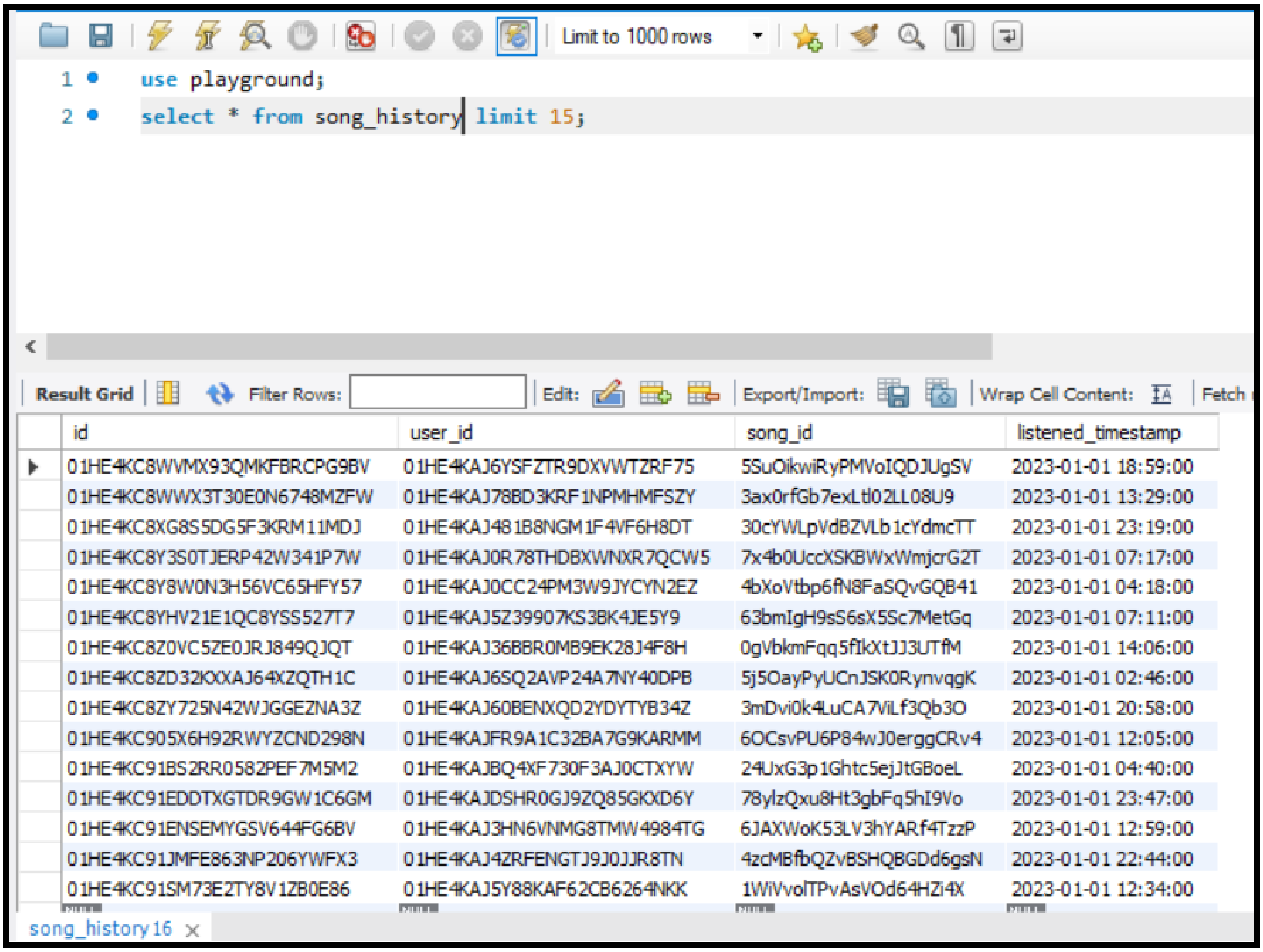
Count of rows for this table:



```sql
use playground;
select count(*) as song_history_table_count from song_history;
```

song_history_table_count
1000

Top 15 rows in the table:



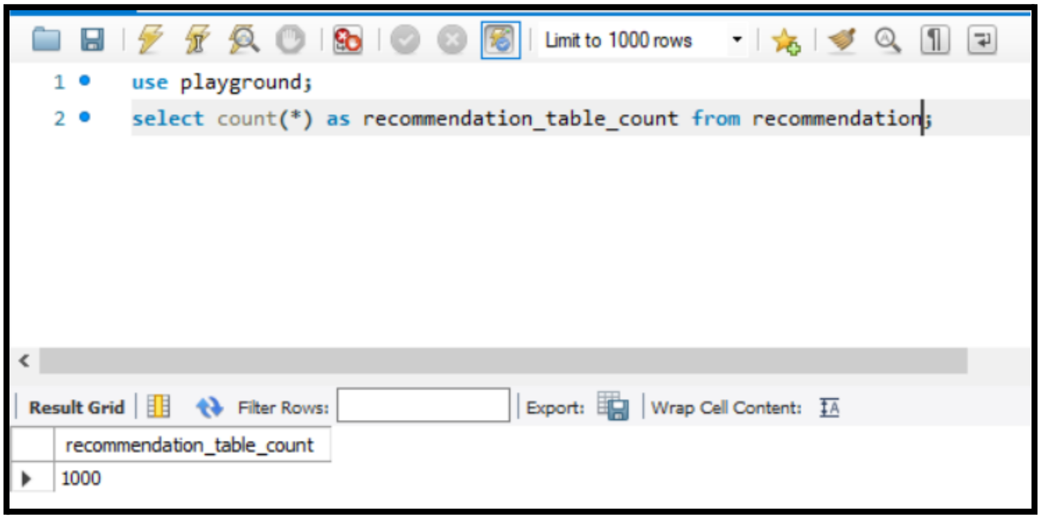
```sql
use playground;
select * from song_history limit 15;
```

id	user_id	song_id	listened_timestamp
01HE4KC8WVMX93QMKFBRCPG9BV	01HE4KAJ6YSFZTR9DXVWTZRF75	5SuOikwiRyPMVoIQDJUgSV	2023-01-01 18:59:00
01HE4KC8WWX3T30E0N6748MZFW	01HE4KAJ78BD3KRF1NPMHMFSZY	3ax0rfGb7exLtl02LL08U9	2023-01-01 13:29:00
01HE4KC8XG8S5DG5F3KRM11MDJ	01HE4KAJ481B8NGM1F4VF6H8DT	30cYWLpVdBZVLb1cYdmcTT	2023-01-01 23:19:00
01HE4KC8Y3S0TJERP42W341P7W	01HE4KAJ0R78THDBXWNXR7QCW5	7x4b0UccXSKBWxWmjcrG2T	2023-01-01 07:17:00
01HE4KC8Y8W0N3H56VC65HFY57	01HE4KAJ0CC24PM3W9JYCYN2EZ	4bXoVtbp6fN8FaSQvGQB41	2023-01-01 04:18:00
01HE4KC8YHV21E1QC8YSS527T7	01HE4KAJ5Z39907KS3BK4JE5Y9	63bmIgH9sS6sX5Sc7MetGq	2023-01-01 07:11:00
01HE4KC8Z0VC5ZE0JRJ849QJQT	01HE4KAJ36BBR0MB9EK28J4F8H	0gVbkmFqq5fIkXtJJ3UTfM	2023-01-01 14:06:00
01HE4KC8ZD32KXXAJ64XZQTH1C	01HE4KAJ6SQ2AVP24A7NY40DPB	5j5OayPyUCnJSK0RynvqgK	2023-01-01 02:46:00
01HE4KC8ZY725N42WJGGEZNA3Z	01HE4KAJ60BENXQD2YDYTYB34Z	3mDvi0k4LuCA7ViLf3Qb3O	2023-01-01 20:58:00
01HE4KC905X6H92RWYZCND298N	01HE4KAJFR9A1C32BA7G9KARMM	6OCsvPU6P84wJ0erggCRv4	2023-01-01 12:05:00
01HE4KC91BS2RR0582PEF7M5M2	01HE4KAJBQ4XF730F3AJ0CTXYW	24UxG3p1Ghtc5ejJtGBoeL	2023-01-01 04:40:00
01HE4KC91EDDTXGTDR9GW1C6GM	01HE4KAJDSHR0GJ9ZQ85GKXD6Y	78ylzQxu8Ht3gbFq5hI9Vo	2023-01-01 23:47:00
01HE4KC91ENSEMYGSV644FG6BV	01HE4KAJ3HN6VNMG8TMW4984TG	6JAXWoK53LV3hYARf4TzzP	2023-01-01 12:59:00
01HE4KC91JMFE863NP206YWFX3	01HE4KAJ4ZRFENGTJ9J0JJR8TN	4zcMBfbQZvBSHQBGDd6gsN	2023-01-01 22:44:00
01HE4KC91SM73E2TY8V1ZB0E86	01HE4KAJ5Y88KAF62CB6264NKK	1WiVvolTPvAsVOd64HZi4X	2023-01-01 12:34:00

f. Table- <u>recommendation:</u>

```
CREATE TABLE recommendation(
user_id VARCHAR(100),
song_id VARCHAR(100),
recommended_timsetamp TIMESTAMP,
PRIMARY KEY (user_id, song_id),
FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE,
FOREIGN KEY (song_id) REFERENCES song(id) ON DELETE CASCADE);
```

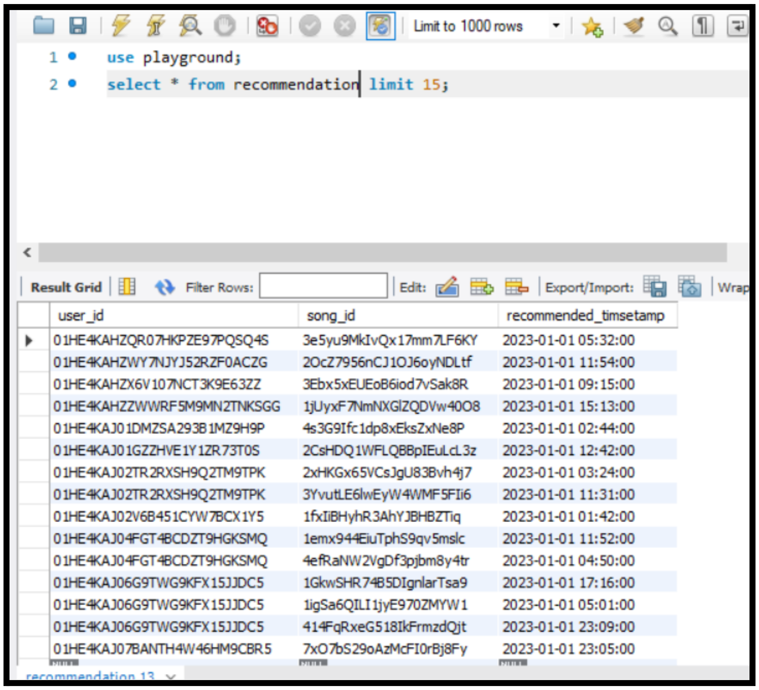
Count of rows for this table:



```
use playground;
select count(*) as recommendation_table_count from recommendation;
```

recommendation_table_count
1000

Top 15 rows in the table:



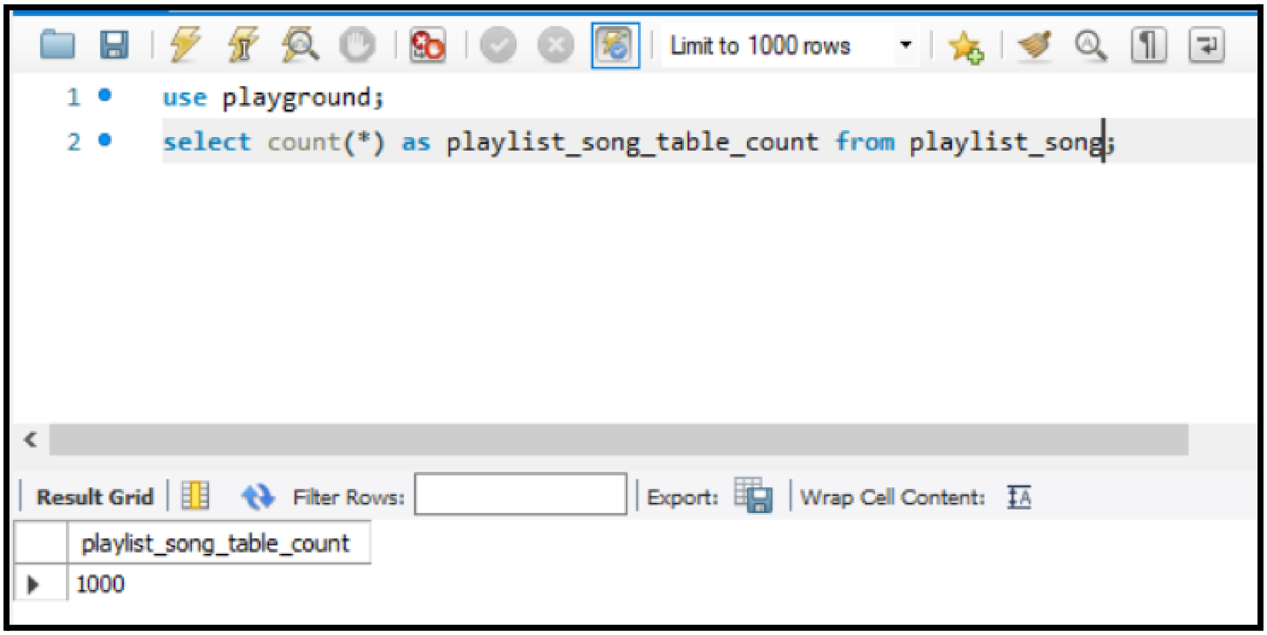
```
use playground;
select * from recommendation limit 15;
```

user_id	song_id	recommended_timsetamp
01HE4KAHZQR07HKPZE97PQSQ4S	3e5yu9MkIvQx17mm7LF6KY	2023-01-01 05:32:00
01HE4KAHZWY7NJYJ52RZF0ACZG	2OcZ7956nCJ1OJ6oyNDLtf	2023-01-01 11:54:00
01HE4KAHZX6V107NCT3K9E63ZZ	3Ebx5xEUEoB6iod7vSak8R	2023-01-01 09:15:00
01HE4KAHZZWWRF5M9MN2TNKSGG	1jUyxF7NmNXGlZQDVw40O8	2023-01-01 15:13:00
01HE4KAJ01DMZSA293B1MZ9H9P	4s3G9Ifc1dp8xEksZxNe8P	2023-01-01 02:44:00
01HE4KAJ01GZZHVE1Y1ZR73T0S	2CsHDQ1WFLQBBpIEuLcL3z	2023-01-01 12:42:00
01HE4KAJ02TR2RXSH9Q2TM9TPK	2xHKGx65VCsJgU83Bvh4j7	2023-01-01 03:24:00
01HE4KAJ02TR2RXSH9Q2TM9TPK	3YvutLE6lwEyW4WMF5FIi6	2023-01-01 11:31:00
01HE4KAJ02V6B451CYW7BCX1Y5	1fxIiBHyhR3AhYJBHBZTiq	2023-01-01 01:42:00
01HE4KAJ04FGT4BCDZT9HGKSMQ	1emx944EiuTphS9qv5mslc	2023-01-01 11:52:00
01HE4KAJ04FGT4BCDZT9HGKSMQ	4efRaNW2VgDf3pjbm8y4tr	2023-01-01 04:50:00
01HE4KAJ06G9TWG9KFX15JJDC5	1GkwSHR74B5DIgnlarTsa9	2023-01-01 17:16:00
01HE4KAJ06G9TWG9KFX15JJDC5	1igSa6QILI1jyE970ZMYW1	2023-01-01 05:01:00
01HE4KAJ06G9TWG9KFX15JJDC5	414FqRxeG518IkFrmzdQjt	2023-01-01 23:09:00
01HE4KAJ07BANTH4W46HM9CBR5	7xO7bS29oAzMcFI0rBj8Fy	2023-01-01 23:05:00

g. Table playlist_song

```
CREATE TABLE playlist_song (
song_id VARCHAR(22),
playlist_id VARCHAR(100),
inserted_timestamp TIMESTAMP,
PRIMARY KEY (song_id, playlist_id),
FOREIGN KEY (playlist_id) REFERENCES playlist(id) ON DELETE CASCADE,
FOREIGN KEY (song_id) REFERENCES song(id) ON DELETE CASCADE);
```

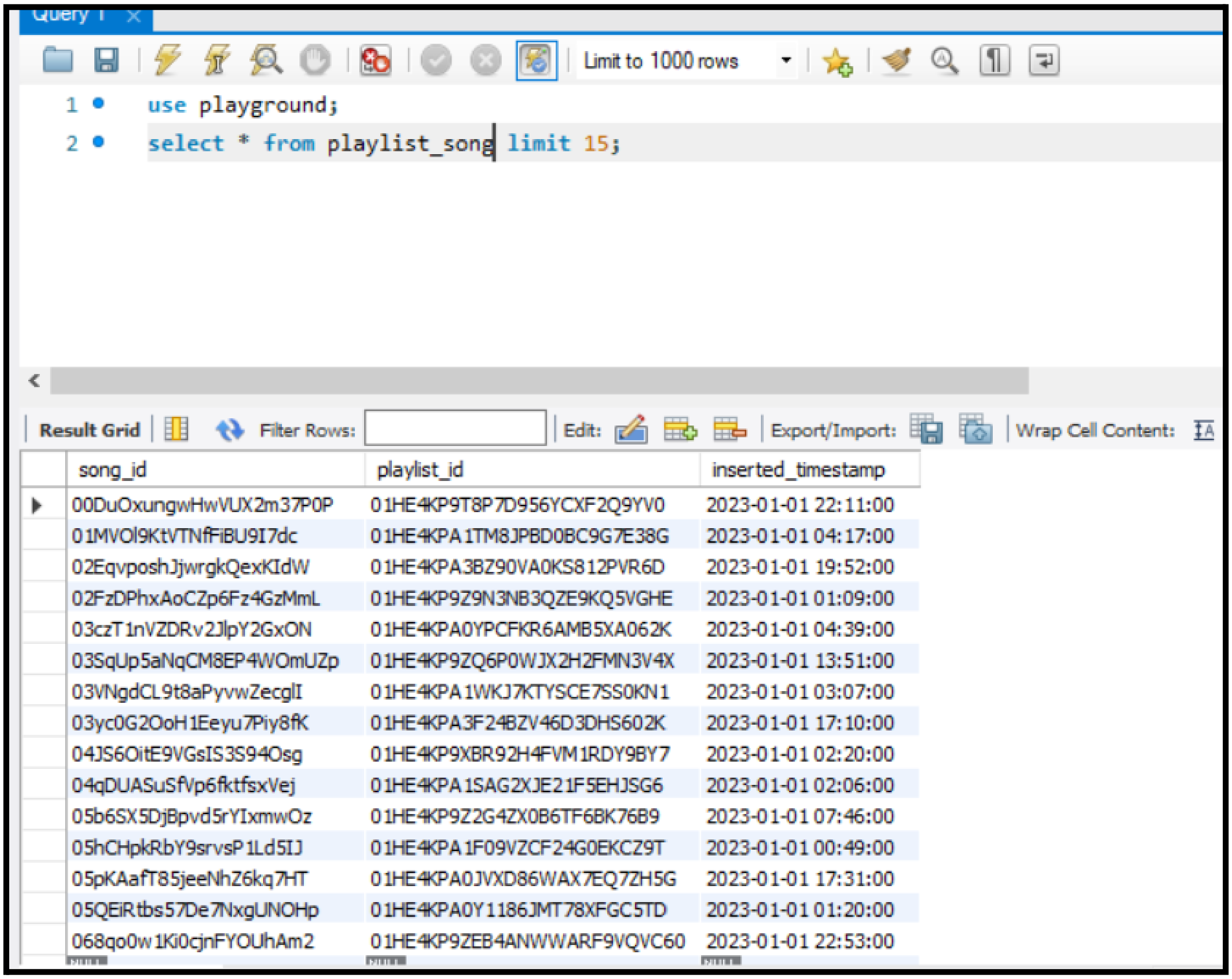
Count of rows for this table:



```
use playground;
select count(*) as playlist_song_table_count from playlist_song;
```

```
playlist_song_table_count
1000
```

Top 15 rows in the table:



Limit to 1000 rows

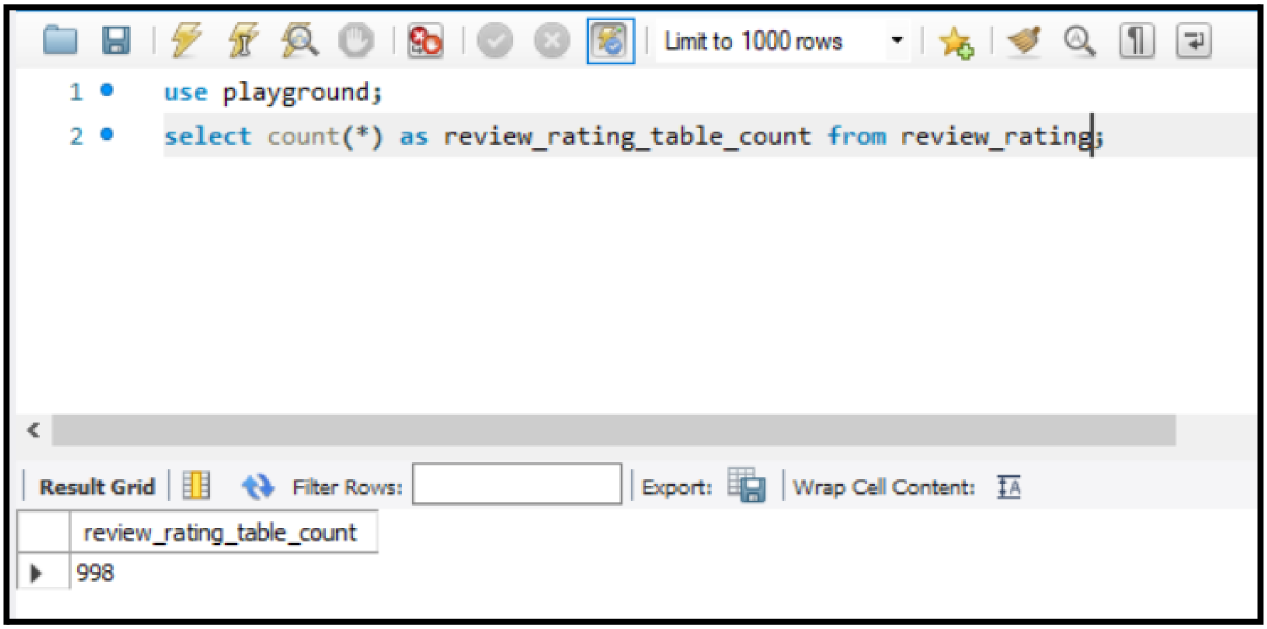
```
use playground;
select * from playlist_song limit 15;
```

song_id	playlist_id	inserted_timestamp
00DuOxungwHwVUX2m37P0P	01HE4KP9T8P7D956YCXF2Q9YV0	2023-01-01 22:11:00
01MVOl9KtVTNfFiBU9I7dc	01HE4KPA1TM8JPBD0BC9G7E38G	2023-01-01 04:17:00
02EqvposhJjwrgkQexKIdW	01HE4KPA3BZ90VA0KS812PVR6D	2023-01-01 19:52:00
02FzDPhxAoCZp6Fz4GzMmL	01HE4KP9Z9N3NB3QZE9KQ5VGHE	2023-01-01 01:09:00
03czT1nVZDRv2JlpY2GxON	01HE4KPA0YPCFKR6AMB5XA062K	2023-01-01 04:39:00
03SqUp5aNqCM8EP4WOmUZp	01HE4KP9ZQ6P0WJX2H2FMN3V4X	2023-01-01 13:51:00
03VNgdCL9t8aPyvwZecglI	01HE4KPA1WKJ7KTYSCE7SS0KN1	2023-01-01 03:07:00
03yc0G2OoH1Eeyu7Piy8fK	01HE4KPA3F24BZV46D3DHS602K	2023-01-01 17:10:00
04JS6OitE9VGsIS3S94Osg	01HE4KP9XBR92H4FVM1RDY9BY7	2023-01-01 02:20:00
04qDUASuSfVp6fktfsxVej	01HE4KPA1SAG2XJE21F5EHJSG6	2023-01-01 02:06:00
05b6SX5DjBpvd5rYIxmwOz	01HE4KP9Z2G4ZX0B6TF6BK76B9	2023-01-01 07:46:00
05hCHpkRbY9srvsP1Ld5IJ	01HE4KPA1F09VZCF24G0EKCZ9T	2023-01-01 00:49:00
05pKAafT85jeeNhZ6kq7HT	01HE4KPA0JVXD86WAX7EQ7ZH5G	2023-01-01 17:31:00
05QEiRtbs57De7NxgUNOHp	01HE4KPA0Y1186JMT78XFGC5TD	2023-01-01 01:20:00
068qo0w1Ki0cjnFYOUhAm2	01HE4KP9ZEB4ANWWARF9VQVC60	2023-01-01 22:53:00

h. Table- review_rating:

CREATE TABLE review_rating(
review_rating INT,
review_id VARCHAR(100),
user_id VARCHAR(100),
PRIMARY KEY (review_id, user_id),
FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE,
FOREIGN KEY (review_id) REFERENCES review(id) ON DELETE CASCADE);

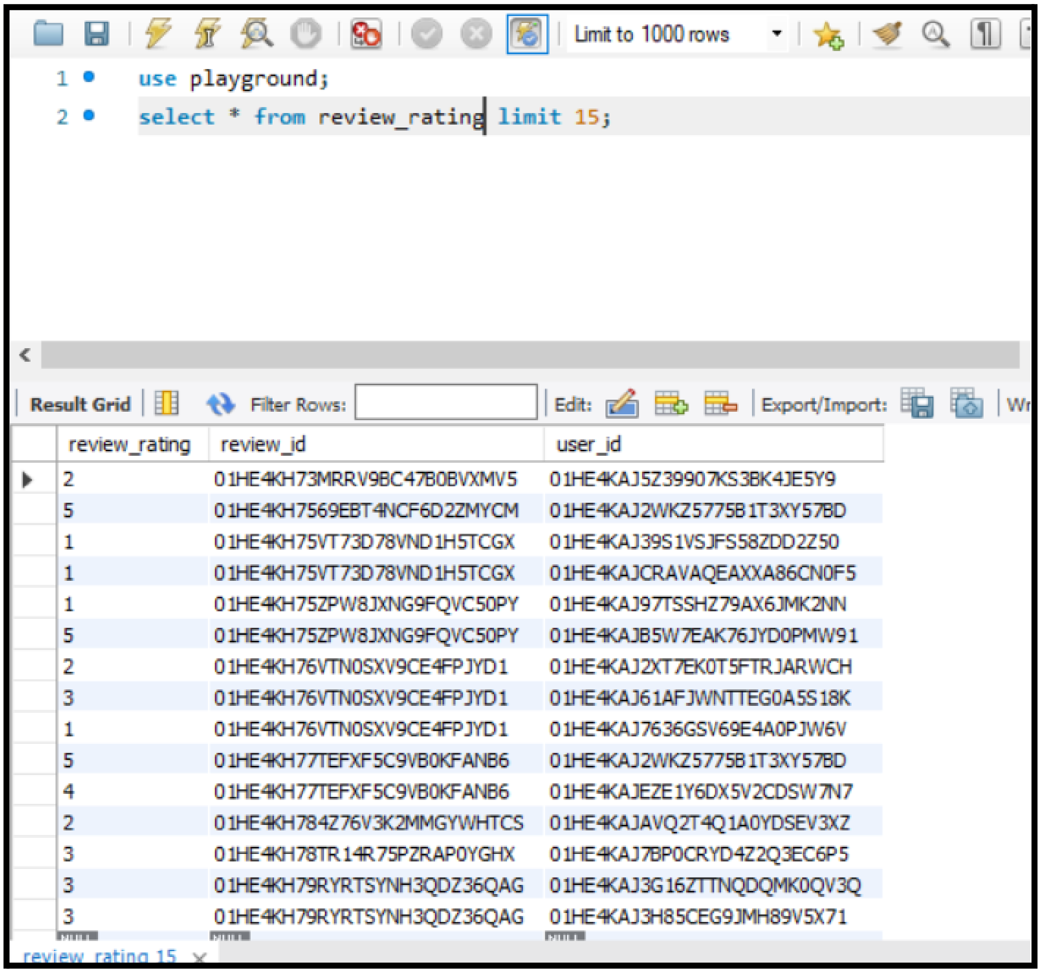
Count of rows for this table:



```sql
use playground;
select count(*) as review_rating_table_count from review_rating;
```

	review_rating_table_count
▶	998

Top 15 rows in the table:



```sql
use playground;
select * from review_rating limit 15;
```

	review_rating	review_id	user_id
▶	2	01HE4KH73MRRV9BC47B0BVXMV5	01HE4KAJ5Z39907KS3BK4JE5Y9
	5	01HE4KH7569EBT4NCF6D2ZMYCM	01HE4KAJ2WKZ5775B1T3XY57BD
	1	01HE4KH75VT73D78VND1H5TCGX	01HE4KAJ39S1VSJFS58ZDD2Z50
	1	01HE4KH75VT73D78VND1H5TCGX	01HE4KAJCRAVAQEAXXA86CN0F5
	1	01HE4KH75ZPW8JXNG9FQVC50PY	01HE4KAJ97TSSHZ79AX6JMK2NN
	5	01HE4KH75ZPW8JXNG9FQVC50PY	01HE4KAJB5W7EAK76JYD0PMW91
	2	01HE4KH76VTN0SXV9CE4FPJYD1	01HE4KAJ2XT7EK0T5FTRJARWCH
	3	01HE4KH76VTN0SXV9CE4FPJYD1	01HE4KAJ61AFJWNTTEG0A5S18K
	1	01HE4KH76VTN0SXV9CE4FPJYD1	01HE4KAJ7636GSV69E4A0PJW6V
	5	01HE4KH77TEFXF5C9VB0KFANB6	01HE4KAJ2WKZ5775B1T3XY57BD
	4	01HE4KH77TEFXF5C9VB0KFANB6	01HE4KAJEZE1Y6DX5V2CDSW7N7
	2	01HE4KH784Z76V3K2MMGYWHTCS	01HE4KAJAVQ2T4Q1A0YDSEV3XZ
	3	01HE4KH78TR14R75PZRAP0YGHX	01HE4KAJ7BP0CRYD4Z2Q3EC6P5
	3	01HE4KH79RYRTSYNH3QDZ36QAG	01HE4KAJ3G16ZTTNQDQMK0QV3Q
	3	01HE4KH79RYRTSYNH3QDZ36QAG	01HE4KAJ3H85CEG9JMH89V5X71

3. Advanced SQL queries implementation:

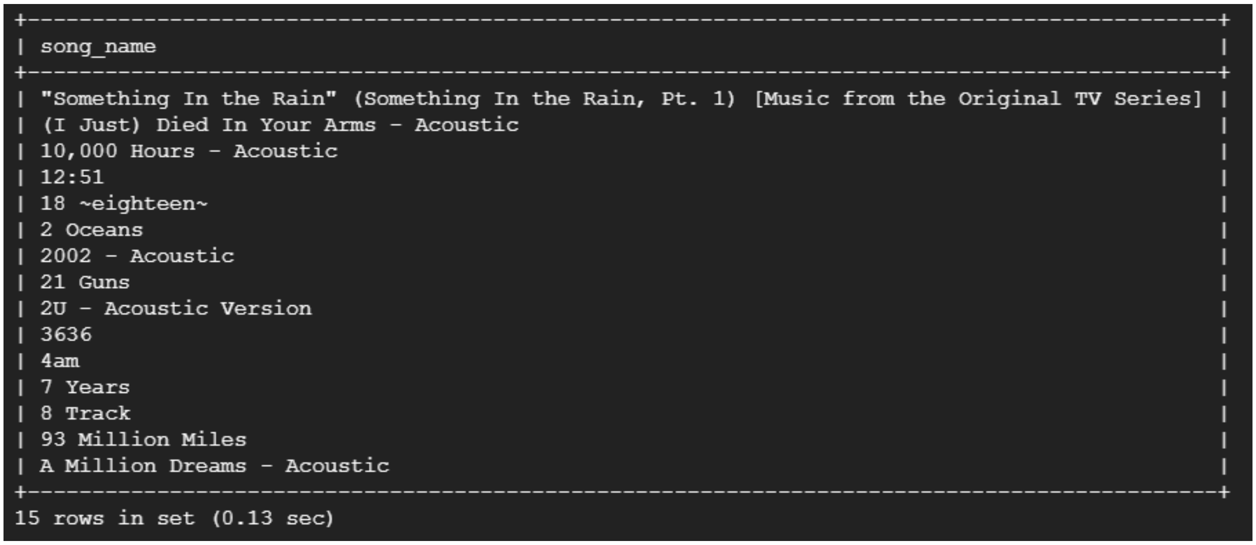
We developed the following advanced SQL queries and analyzed the performances by indexing different attributes involved in the query

a. Query 1

The following query is used to generate song recommendations for a particular user based on their listening history within the same genre.
The query uses cosine similarity metric to identify similar songs and also filters out already listened songs from the recommendation list.

```
SELECT DISTINCT b.song_name FROM (SELECT
         a.song_name,
         Round((a.popularity*rec.popularity + a.danceability*rec.danceability +
a.energy*rec.energy + a.loudness*rec.loudness + a.mode*rec.mode +
a.speechiness*rec.speechiness + a.acousticness*rec.acousticness +
a.instrumentalness*rec.instrumentalness + a.liveness*rec.liveness +
a.valence*rec.valence + a.tempo*rec.tempo)/ (Sqrt(Power(Truncate(a.popularity,
2), 2) + Power(Truncate(a.danceability, 2), 2) + Power(Truncate(a.energy, 2),
2) + Power(Truncate(a.loudness, 2), 2) + Power(Truncate(a.mode, 2), 2) +
Power(Truncate(a.speechiness, 2), 2) + Power(Truncate(a.acousticness, 2), 2) +
Power(Truncate(a.instrumentalness, 2), 2) + Power(Truncate(a.liveness, 2), 2) +
Power(Truncate(a.valence, 2), 2) + Power(Truncate(a.tempo, 2), 2)) *
Sqrt(Power(Truncate(rec.popularity, 2), 2) + Power(Truncate(rec.danceability,
2), 2) + Power(Truncate(rec.energy, 2), 2) + Power(Truncate(rec.loudness, 2),
2) + Power(Truncate(rec.mode, 2), 2) + Power(Truncate(rec.speechiness, 2), 2) +
Power(Truncate(rec.acousticness, 2), 2) + Power(Truncate(rec.instrumentalness,
2), 2) + Power(Truncate(rec.liveness, 2), 2) + Power(Truncate(rec.valence, 2),
2) + Power(Truncate(rec.tempo, 2), 2))),2) cosine_sim
  FROM song a
  JOIN
    (SELECT s.* FROM song s JOIN(
      SELECT DISTINCT song_id, listened_timestamp
      FROM song_history
      WHERE user_id = "01he4kaj37dam35h94genfkagz"
      ORDER BY listened_timestamp) AS r
  ON s.id = r.song_id LIMIT 3) AS rec ON a.song_genre = rec.song_genre
  WHERE a.id NOT IN (
  SELECT id
     FROM song
     WHERE id IN
     (SELECT DISTINCT song_id
          FROM song_history
          WHERE user_id = "01he4kaj37dam35h94genfkagz"))) AS b
ORDER BY song_name
LIMIT 15;
```

Results:



```
+--------------------------------------------------------------------------------------------+
| song_name                                                                                  |
+--------------------------------------------------------------------------------------------+
| "Something In the Rain" (Something In the Rain, Pt. 1) [Music from the Original TV Series] |
| (I Just) Died In Your Arms - Acoustic                                                      |
| 10,000 Hours - Acoustic                                                                    |
| 12:51                                                                                      |
| 18 ~eighteen~                                                                              |
| 2 Oceans                                                                                   |
| 2002 - Acoustic                                                                            |
| 21 Guns                                                                                    |
| 2U - Acoustic Version                                                                      |
| 3636                                                                                       |
| 4am                                                                                        |
| 7 Years                                                                                    |
| 8 Track                                                                                    |
| 93 Million Miles                                                                           |
| A Million Dreams - Acoustic                                                                |
+--------------------------------------------------------------------------------------------+
15 rows in set (0.13 sec)
```

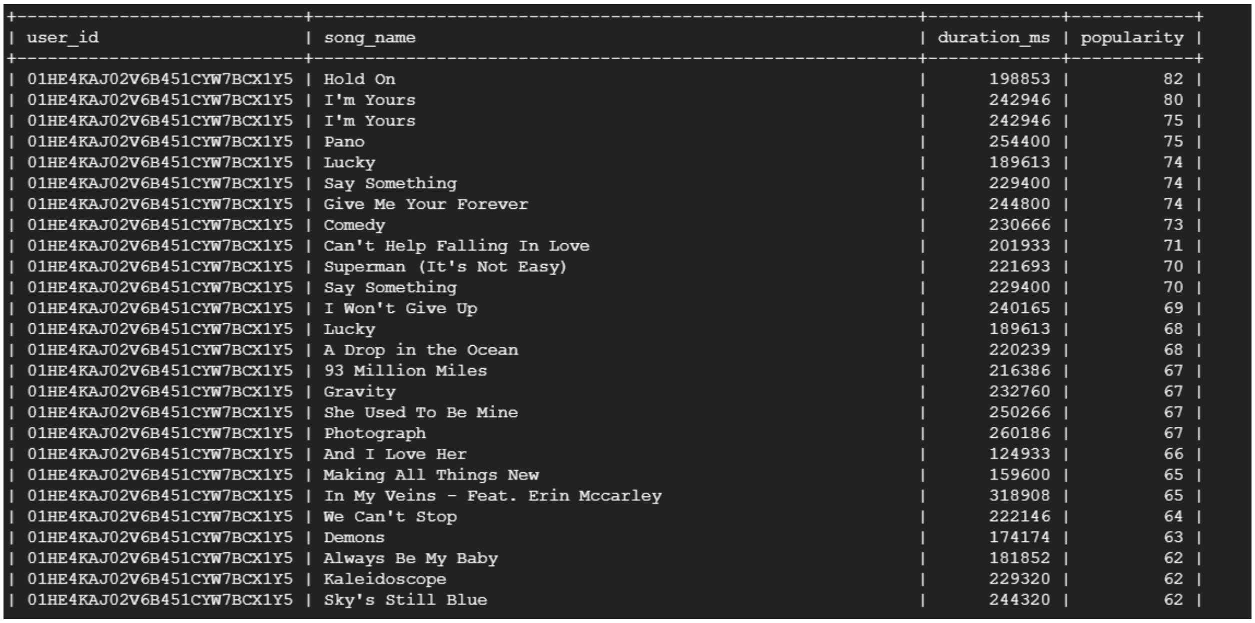
b. Query 2

The following query is used to get a list of songs that a user has listened but has not yet added to any of his playlists. This query also analyses whether the song is explicit or not and also sorts the result by popularity in decreasing order and the song duration in increasing order. Explicit content is only suggested to users greater

```
SELECT
    sh.user_id,
    s.song_name,
    s.duration_ms,
    s.popularity
  FROM
    song_history sh
    JOIN song s ON sh.song_id = s.id
    JOIN USER u ON sh.user_id = u.id
  WHERE (((s.explicit = 1 OR s.explicit = 0 ) AND u.age >= 25) OR (s.explicit =
0 AND u.age < 25))
  AND  sh.user_id = '01HE4KAJ02V6B451CYW7BCX1Y5'
  EXCEPT
  SELECT
    p.created_by_user,
    s.song_name,
  s.duration_ms,
  s.popularity
 FROM
  playlist_song ps
  JOIN playlist p ON ps.playlist_id = p.id
  JOIN song s ON ps.song_id = s.id
  JOIN USER u ON p.created_by_user = u.id

 WHERE
   p.created_by_user = '01HE4KAJ02V6B451CYW7BCX1Y5'
   AND (((s.explicit = 1 OR s.explicit = 0 ) AND u.age >= 25) OR (s.explicit
= 0 AND u.age < 25))
 ORDER BY popularity DESC, duration_ms;
```

Output: Truncated too large

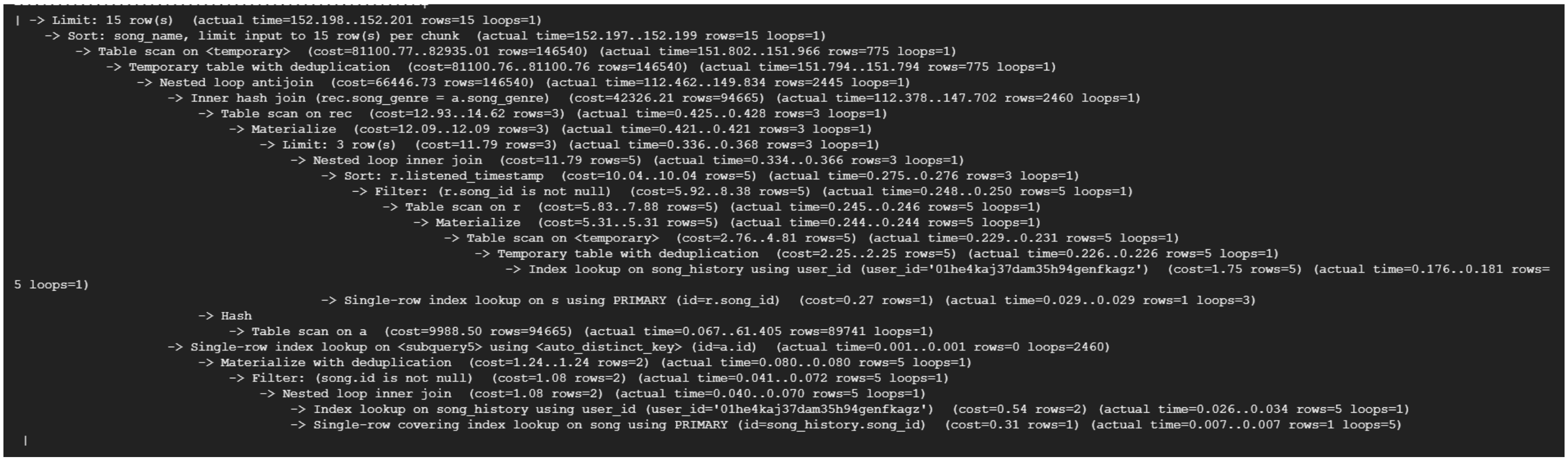


```
+----------------------------+------------------------------------------+-------------+------------+
| user_id                    | song_name                                | duration_ms | popularity |
+----------------------------+------------------------------------------+-------------+------------+
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Hold On                                  |      198853 |         82 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | I'm Yours                                |      242946 |         80 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | I'm Yours                                |      242946 |         75 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Pano                                     |      254400 |         75 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Lucky                                    |      189613 |         74 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Say Something                            |      229400 |         74 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Give Me Your Forever                     |      244800 |         74 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Comedy                                   |      230666 |         73 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Can't Help Falling In Love               |      201933 |         71 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Superman (It's Not Easy)                 |      221693 |         70 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Say Something                            |      229400 |         70 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | I Won't Give Up                          |      240165 |         69 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Lucky                                    |      189613 |         68 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | A Drop in the Ocean                      |      220239 |         68 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | 93 Million Miles                         |      216386 |         67 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Gravity                                  |      232760 |         67 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | She Used To Be Mine                      |      250266 |         67 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Photograph                               |      260186 |         67 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | And I Love Her                           |      124933 |         66 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Making All Things New                    |      159600 |         65 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | In My Veins - Feat. Erin Mccarley        |      318908 |         65 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | We Can't Stop                            |      222146 |         64 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Demons                                   |      174174 |         63 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Always Be My Baby                        |      181852 |         62 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Kaleidoscope                             |      229320 |         62 |
| 01HE4KAJ02V6B451CYW7BCX1Y5 | Sky's Still Blue                         |      244320 |         62 |
```

Indexing Analysis

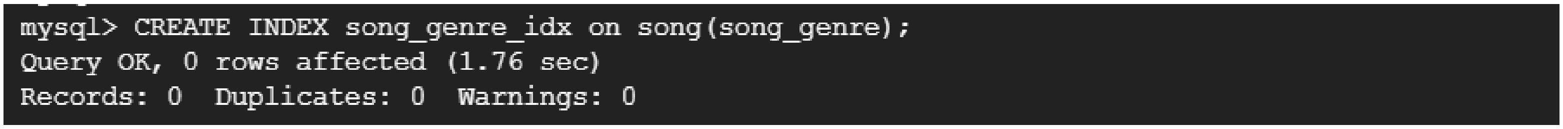
Query 1:

EXPLAIN ANALYSE on regular query



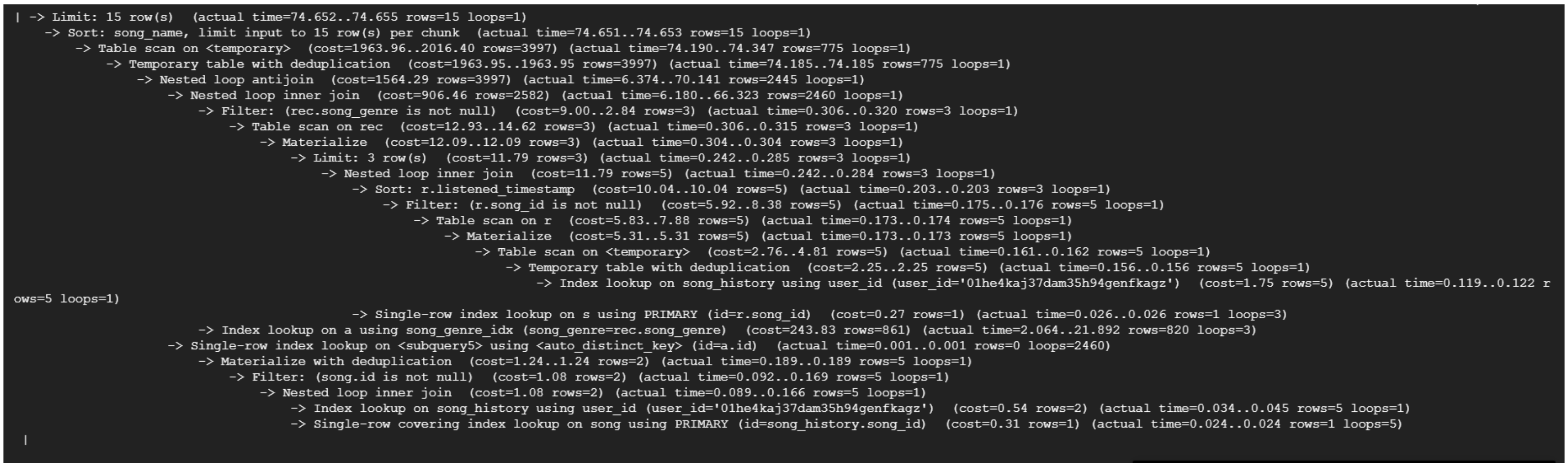
```
| -> Limit: 15 row(s)  (actual time=152.198..152.201 rows=15 loops=1)
    -> Sort: song_name, limit input to 15 row(s) per chunk  (actual time=152.197..152.199 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=81100.77..82935.01 rows=146540) (actual time=151.802..151.966 rows=775 loops=1)
            -> Temporary table with deduplication  (cost=81100.76..81100.76 rows=146540) (actual time=151.794..151.794 rows=775 loops=1)
                -> Nested loop antijoin  (cost=66446.73 rows=146540) (actual time=112.462..149.834 rows=2445 loops=1)
                    -> Inner hash join (rec.song_genre = a.song_genre)  (cost=42326.21 rows=94665) (actual time=112.378..147.702 rows=2460 loops=1)
                        -> Table scan on rec  (cost=12.93..14.62 rows=3) (actual time=0.425..0.428 rows=3 loops=1)
                            -> Materialize  (cost=12.09..12.09 rows=3) (actual time=0.421..0.421 rows=3 loops=1)
                                -> Limit: 3 row(s)  (cost=11.79 rows=3) (actual time=0.336..0.368 rows=3 loops=1)
                                    -> Nested loop inner join  (cost=11.79 rows=5) (actual time=0.334..0.366 rows=3 loops=1)
                                        -> Sort: r.listened_timestamp  (cost=10.04..10.04 rows=5) (actual time=0.275..0.276 rows=3 loops=1)
                                            -> Filter: (r.song_id is not null)  (cost=5.92..8.38 rows=5) (actual time=0.248..0.250 rows=5 loops=1)
                                                -> Table scan on r  (cost=5.83..7.88 rows=5) (actual time=0.245..0.246 rows=5 loops=1)
                                                    -> Materialize  (cost=5.31..5.31 rows=5) (actual time=0.244..0.244 rows=5 loops=1)
                                                        -> Table scan on <temporary>  (cost=2.76..4.81 rows=5) (actual time=0.229..0.231 rows=5 loops=1)
                                                            -> Temporary table with deduplication  (cost=2.25..2.25 rows=5) (actual time=0.226..0.226 rows=5 loops=1)
                                                                -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=1.75 rows=5) (actual time=0.176..0.181 rows=
5 loops=1)
                                        -> Single-row index lookup on s using PRIMARY (id=r.song_id)  (cost=0.27 rows=1) (actual time=0.029..0.029 rows=1 loops=3)
                        -> Hash
                            -> Table scan on a  (cost=9988.50 rows=94665) (actual time=0.067..61.405 rows=89741 loops=1)
                    -> Single-row index lookup on <subquery5> using <auto_distinct_key> (id=a.id)  (actual time=0.001..0.001 rows=0 loops=2460)
                        -> Materialize with deduplication  (cost=1.24..1.24 rows=2) (actual time=0.080..0.080 rows=5 loops=1)
                            -> Filter: (song.id is not null)  (cost=1.08 rows=2) (actual time=0.041..0.072 rows=5 loops=1)
                                -> Nested loop inner join  (cost=1.08 rows=2) (actual time=0.040..0.070 rows=5 loops=1)
                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=0.54 rows=2) (actual time=0.026..0.034 rows=5 loops=1)
                                    -> Single-row covering index lookup on song using PRIMARY (id=song_history.song_id)  (cost=0.31 rows=1) (actual time=0.007..0.007 rows=1 loops=5)
 |
```

Adding index on song_genre in song table as it is used in the join condition for generating song recommendations within the same genre. (Deleting index after **EXPLAIN ANALYSE**)



```
mysql> CREATE INDEX song_genre_idx on song(song_genre);
Query OK, 0 rows affected (1.76 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

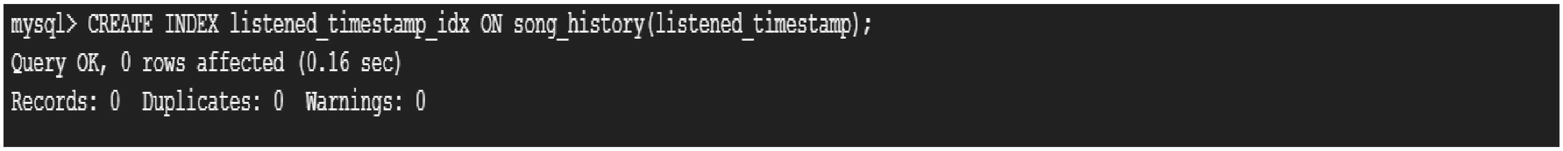
Running Explain Analyze:



```
| -> Limit: 15 row(s)  (actual time=74.652..74.655 rows=15 loops=1)
    -> Sort: song_name, limit input to 15 row(s) per chunk  (actual time=74.651..74.653 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=1963.96..2016.40 rows=3997) (actual time=74.190..74.347 rows=775 loops=1)
            -> Temporary table with deduplication  (cost=1963.95..1963.95 rows=3997) (actual time=74.185..74.185 rows=775 loops=1)
                -> Nested loop antijoin  (cost=1564.29 rows=3997) (actual time=6.374..70.141 rows=2445 loops=1)
                    -> Nested loop inner join  (cost=906.46 rows=2582) (actual time=6.180..66.323 rows=2460 loops=1)
                        -> Filter: (rec.song_genre is not null)  (cost=9.00..2.84 rows=3) (actual time=0.306..0.320 rows=3 loops=1)
                            -> Table scan on rec  (cost=12.93..14.62 rows=3) (actual time=0.306..0.315 rows=3 loops=1)
                                -> Materialize  (cost=12.09..12.09 rows=3) (actual time=0.304..0.304 rows=3 loops=1)
                                    -> Limit: 3 row(s)  (cost=11.79 rows=3) (actual time=0.242..0.285 rows=3 loops=1)
                                        -> Nested loop inner join  (cost=11.79 rows=5) (actual time=0.242..0.284 rows=3 loops=1)
                                            -> Sort: r.listened_timestamp  (cost=10.04..10.04 rows=5) (actual time=0.203..0.203 rows=3 loops=1)
                                                -> Filter: (r.song_id is not null)  (cost=5.92..8.38 rows=5) (actual time=0.175..0.176 rows=5 loops=1)
                                                    -> Table scan on r  (cost=5.83..7.88 rows=5) (actual time=0.173..0.174 rows=5 loops=1)
                                                        -> Materialize  (cost=5.31..5.31 rows=5) (actual time=0.173..0.173 rows=5 loops=1)
                                                            -> Table scan on <temporary>  (cost=2.76..4.81 rows=5) (actual time=0.161..0.162 rows=5 loops=1)
                                                                -> Temporary table with deduplication  (cost=2.25..2.25 rows=5) (actual time=0.156..0.156 rows=5 loops=1)
                                                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=1.75 rows=5) (actual time=0.119..0.122 r
ows=5 loops=1)
                                            -> Single-row index lookup on s using PRIMARY (id=r.song_id)  (cost=0.27 rows=1) (actual time=0.026..0.026 rows=1 loops=3)
                        -> Index lookup on a using song_genre_idx (song_genre=rec.song_genre)  (cost=243.83 rows=861) (actual time=2.064..21.892 rows=820 loops=3)
                    -> Single-row index lookup on <subquery5> using <auto_distinct_key> (id=a.id)  (actual time=0.001..0.001 rows=0 loops=2460)
                        -> Materialize with deduplication  (cost=1.24..1.24 rows=2) (actual time=0.189..0.189 rows=5 loops=1)
                            -> Filter: (song.id is not null)  (cost=1.08 rows=2) (actual time=0.092..0.169 rows=5 loops=1)
                                -> Nested loop inner join  (cost=1.08 rows=2) (actual time=0.089..0.166 rows=5 loops=1)
                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=0.54 rows=2) (actual time=0.034..0.045 rows=5 loops=1)
                                    -> Single-row covering index lookup on song using PRIMARY (id=song_history.song_id)  (cost=0.31 rows=1) (actual time=0.024..0.024 rows=1 loops=5)
 |
```

For index 1 here, we added the index on song_genre column in the song table as this column was used in getting song recommendations using a self join technique. This index was created to make the optimizer use a better joining strategy. This index, as seen in the cost plan, helped us remove the **hash-join** strategy used in the plain execution and it was replaced with a nested loop inner join and a filter condition. The cost for the join section dropped from (cost = 42326.21 rows = 94665) to (cost = 906.46 rows = 2582)**.** The overall runtime also reduced by around 50%.

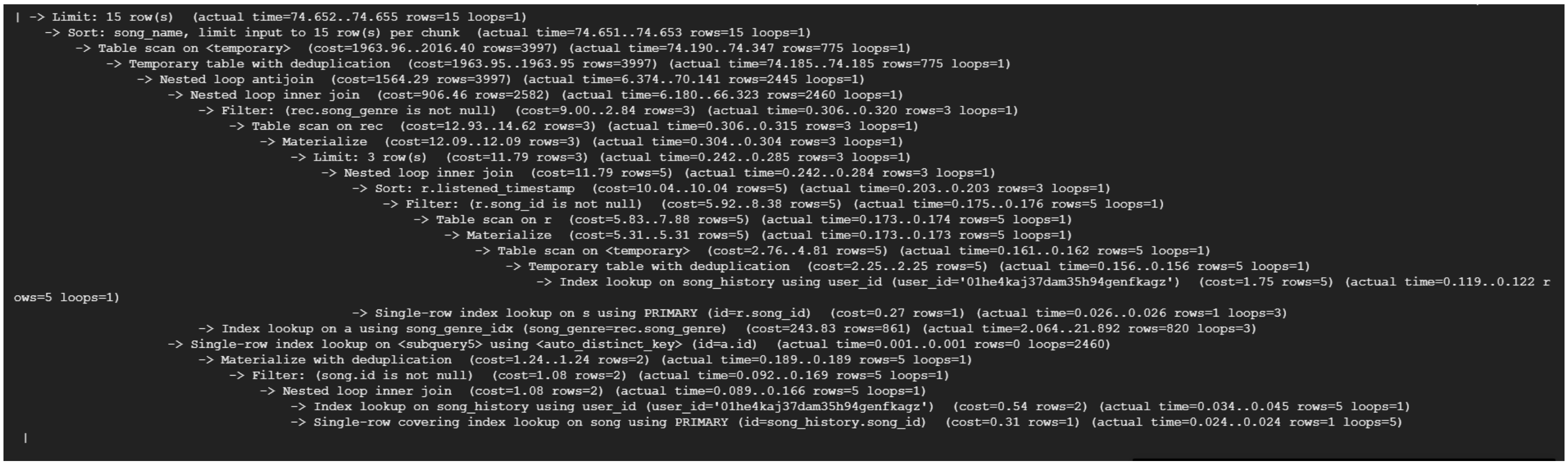
Adding index on listened_timestamp in song_history table as this attribute is used to get only top 3 recently listened song for generating recommendations. (Deleting index after **EXPLAIN ANALYSE**)



```
mysql> CREATE INDEX listened_timestamp_idx ON song_history(listened_timestamp);
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

For index 2 here, we added the index on listened_timestamp column from the song_history table as this column was used in retrieving most recently listened 3 songs of a user. We added this index to reduce the sorting time in this process. After naively adding the index we noticed there was no change in the query cost and plan as shown in the output below. We noticed that our query was ordering records using listened_timestamp column OUTSIDE of subquery, due to which the optimizer was not able to use and benefit the index. Hence we change our query structure and PUSHED the listened_timestamp value inside the subquery and ran the explain analyse command again.

Running Explain Analyse

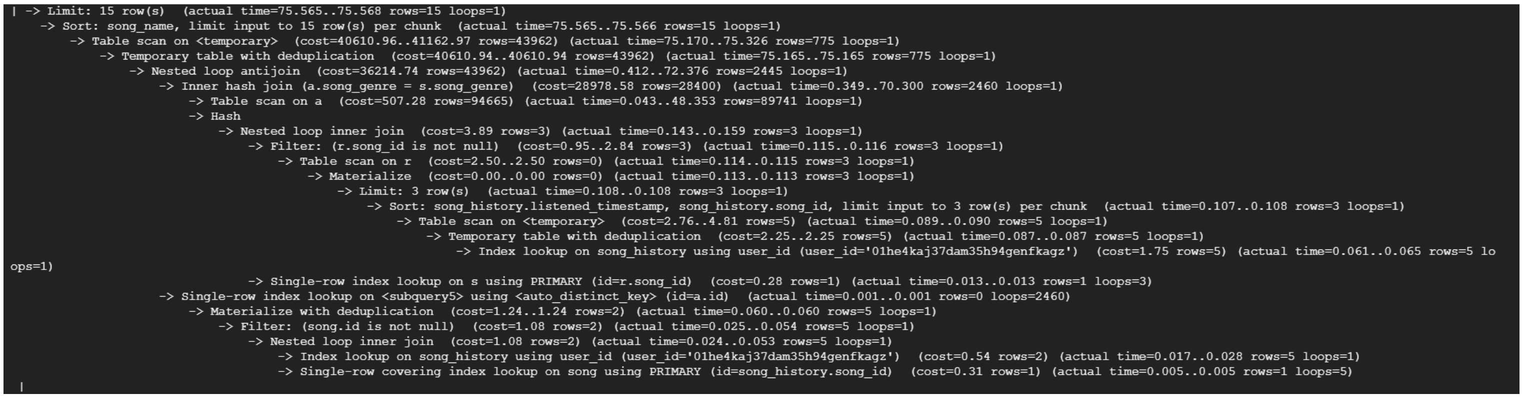


```
| -> Limit: 15 row(s)  (actual time=74.652..74.655 rows=15 loops=1)
    -> Sort: song_name, limit input to 15 row(s) per chunk  (actual time=74.651..74.653 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=1963.96..2016.40 rows=3997) (actual time=74.190..74.347 rows=775 loops=1)
            -> Temporary table with deduplication  (cost=1963.95..1963.95 rows=3997) (actual time=74.185..74.185 rows=775 loops=1)
                -> Nested loop antijoin  (cost=1564.29 rows=3997) (actual time=6.374..70.141 rows=2445 loops=1)
                    -> Nested loop inner join  (cost=906.46 rows=2582) (actual time=6.180..66.323 rows=2460 loops=1)
                        -> Filter: (rec.song_genre is not null)  (cost=9.00..2.84 rows=3) (actual time=0.306..0.320 rows=3 loops=1)
                            -> Table scan on rec  (cost=12.93..14.62 rows=3) (actual time=0.306..0.315 rows=3 loops=1)
                                -> Materialize  (cost=12.09..12.09 rows=3) (actual time=0.304..0.304 rows=3 loops=1)
                                    -> Limit: 3 row(s)  (cost=11.79 rows=3) (actual time=0.242..0.285 rows=3 loops=1)
                                        -> Nested loop inner join  (cost=11.79 rows=5) (actual time=0.242..0.284 rows=3 loops=1)
                                            -> Sort: r.listened_timestamp  (cost=10.04..10.04 rows=5) (actual time=0.203..0.203 rows=3 loops=1)
                                                -> Filter: (r.song_id is not null)  (cost=5.92..8.38 rows=5) (actual time=0.175..0.176 rows=5 loops=1)
                                                    -> Table scan on r  (cost=5.83..7.88 rows=5) (actual time=0.173..0.174 rows=5 loops=1)
                                                        -> Materialize  (cost=5.31..5.31 rows=5) (actual time=0.173..0.173 rows=5 loops=1)
                                                            -> Table scan on <temporary>  (cost=2.76..4.81 rows=5) (actual time=0.161..0.162 rows=5 loops=1)
                                                                -> Temporary table with deduplication  (cost=2.25..2.25 rows=5) (actual time=0.156..0.156 rows=5 loops=1)
                                                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=1.75 rows=5) (actual time=0.119..0.122 r
ows=5 loops=1)
                                            -> Single-row index lookup on s using PRIMARY (id=r.song_id)  (cost=0.27 rows=1) (actual time=0.026..0.026 rows=1 loops=3)
                        -> Index lookup on a using song_genre_idx (song_genre=rec.song_genre)  (cost=243.83 rows=861) (actual time=2.064..21.892 rows=820 loops=3)
                    -> Single-row index lookup on <subquery5> using <auto_distinct_key> (id=a.id)  (actual time=0.001..0.001 rows=0 loops=2460)
                        -> Materialize with deduplication  (cost=1.24..1.24 rows=2) (actual time=0.189..0.189 rows=5 loops=1)
                            -> Filter: (song.id is not null)  (cost=1.08 rows=2) (actual time=0.092..0.169 rows=5 loops=1)
                                -> Nested loop inner join  (cost=1.08 rows=2) (actual time=0.089..0.166 rows=5 loops=1)
                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=0.54 rows=2) (actual time=0.034..0.045 rows=5 loops=1)
                                    -> Single-row covering index lookup on song using PRIMARY (id=song_history.song_id)  (cost=0.31 rows=1) (actual time=0.024..0.024 rows=1 loops=5)
 |
```

Modifying Query and Running Explain Analyse again

Original Query: `(<SUBQUERY>) ORDER BY listened_timestamp`

Modified Query: `(<SUBQUERY> ORDER BY listened_timestamp)`

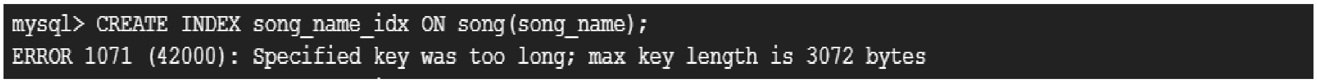


```
| -> Limit: 15 row(s)  (actual time=75.565..75.568 rows=15 loops=1)
    -> Sort: song_name, limit input to 15 row(s) per chunk  (actual time=75.565..75.566 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=40610.96..41162.97 rows=43962) (actual time=75.170..75.326 rows=775 loops=1)
            -> Temporary table with deduplication  (cost=40610.94..40610.94 rows=43962) (actual time=75.165..75.165 rows=775 loops=1)
                -> Nested loop antijoin  (cost=36214.74 rows=43962) (actual time=0.412..72.376 rows=2445 loops=1)
                    -> Inner hash join (a.song_genre = s.song_genre)  (cost=28978.58 rows=28400) (actual time=0.349..70.300 rows=2460 loops=1)
                        -> Table scan on a  (cost=507.28 rows=94665) (actual time=0.043..48.353 rows=89741 loops=1)
                        -> Hash
                            -> Nested loop inner join  (cost=3.89 rows=3) (actual time=0.143..0.159 rows=3 loops=1)
                                -> Filter: (r.song_id is not null)  (cost=0.95..2.84 rows=3) (actual time=0.115..0.116 rows=3 loops=1)
                                    -> Table scan on r  (cost=2.50..2.50 rows=0) (actual time=0.114..0.115 rows=3 loops=1)
                                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=0.113..0.113 rows=3 loops=1)
                                            -> Limit: 3 row(s)  (actual time=0.108..0.108 rows=3 loops=1)
                                                -> Sort: song_history.listened_timestamp, song_history.song_id, limit input to 3 row(s) per chunk  (actual time=0.107..0.108 rows=3 loops=1)
                                                    -> Table scan on <temporary>  (cost=2.76..4.81 rows=5) (actual time=0.089..0.090 rows=5 loops=1)
                                                        -> Temporary table with deduplication  (cost=2.25..2.25 rows=5) (actual time=0.087..0.087 rows=5 loops=1)
                                                            -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=1.75 rows=5) (actual time=0.061..0.065 rows=5 lo
ops=1)
                                -> Single-row index lookup on s using PRIMARY (id=r.song_id)  (cost=0.28 rows=1) (actual time=0.013..0.013 rows=1 loops=3)
                    -> Single-row index lookup on <subquery5> using <auto_distinct_key> (id=a.id)  (actual time=0.001..0.001 rows=0 loops=2460)
                        -> Materialize with deduplication  (cost=1.24..1.24 rows=2) (actual time=0.060..0.060 rows=5 loops=1)
                            -> Filter: (song.id is not null)  (cost=1.08 rows=2) (actual time=0.025..0.054 rows=5 loops=1)
                                -> Nested loop inner join  (cost=1.08 rows=2) (actual time=0.024..0.053 rows=5 loops=1)
                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=0.54 rows=2) (actual time=0.017..0.028 rows=5 loops=1)
                                    -> Single-row covering index lookup on song using PRIMARY (id=song_history.song_id)  (cost=0.31 rows=1) (actual time=0.005..0.005 rows=1 loops=5)
 |
```

After modifying the query and running analyse again, we saw query time was reduced by around 50%, The optimizer used the index on listened_timestamp and the query plan was shortened. The **Materialization** step for storing the sorted 3 records was bypassed leading to reduction in cost from (cost = 12.09) in first Materialization and (cost = 5.31) in second Materialization to (cost = 0.00) in the query that used the listened_timestamp index.

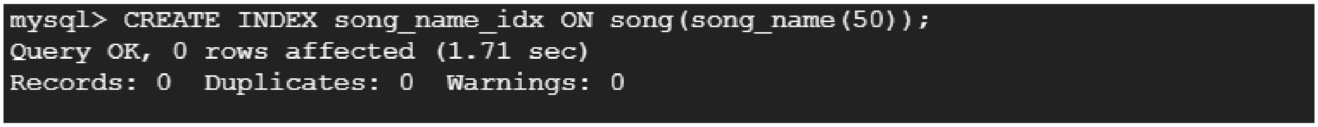
Adding index on song_name in song table as this attribute is used to order the results at the end of the query.

For index 3 here, as this attribute is used to order the results at the end of the query.



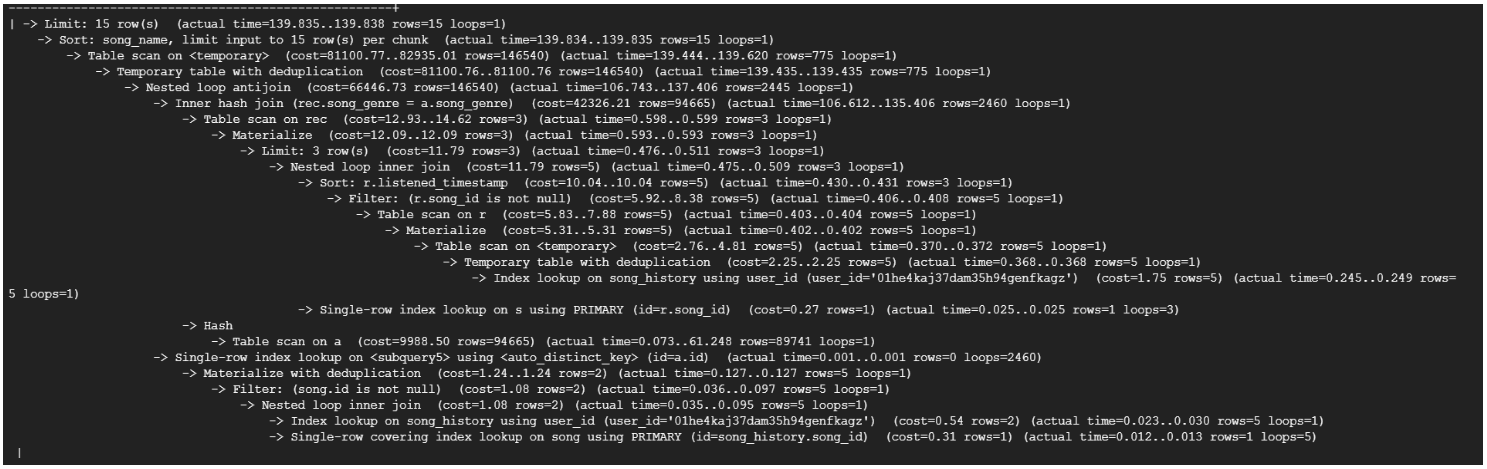
```
mysql> CREATE INDEX song_name_idx ON song(song_name);
ERROR 1071 (42000): Specified key was too long; max key length is 3072 bytes
```

As the length of the song_name field exceeded the maximum key length that can be indexed, we were not able to create an index on the song_name attribute. Hence we tried to create an index on the first 50 characters from the song_name field as these are sufficient for sorting the results.
We also noted that this index took longer time for creation



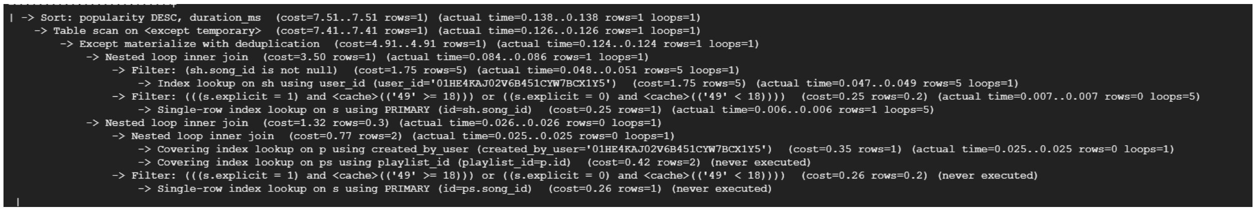
```
mysql> CREATE INDEX song_name_idx ON song(song_name(50));
Query OK, 0 rows affected (1.71 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Running Explain Analyse



```
--------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=139.835..139.838 rows=15 loops=1)
    -> Sort: song_name, limit input to 15 row(s) per chunk  (actual time=139.834..139.835 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=81100.77..82935.01 rows=146540) (actual time=139.444..139.620 rows=775 loops=1)
            -> Temporary table with deduplication  (cost=81100.76..81100.76 rows=146540) (actual time=139.435..139.435 rows=775 loops=1)
                -> Nested loop antijoin  (cost=66446.73 rows=146540) (actual time=106.743..137.406 rows=2445 loops=1)
                    -> Inner hash join (rec.song_genre = a.song_genre)  (cost=42326.21 rows=94665) (actual time=106.612..135.406 rows=2460 loops=1)
                        -> Table scan on rec  (cost=12.93..14.62 rows=3) (actual time=0.598..0.599 rows=3 loops=1)
                            -> Materialize  (cost=12.09..12.09 rows=3) (actual time=0.593..0.593 rows=3 loops=1)
                                -> Limit: 3 row(s)  (cost=11.79 rows=3) (actual time=0.476..0.511 rows=3 loops=1)
                                    -> Nested loop inner join  (cost=11.79 rows=5) (actual time=0.475..0.509 rows=3 loops=1)
                                        -> Sort: r.listened_timestamp  (cost=10.04..10.04 rows=5) (actual time=0.430..0.431 rows=3 loops=1)
                                            -> Filter: (r.song_id is not null)  (cost=5.92..8.38 rows=5) (actual time=0.406..0.408 rows=5 loops=1)
                                                -> Table scan on r  (cost=5.83..7.88 rows=5) (actual time=0.403..0.404 rows=5 loops=1)
                                                    -> Materialize  (cost=5.31..5.31 rows=5) (actual time=0.402..0.402 rows=5 loops=1)
                                                        -> Table scan on <temporary>  (cost=2.76..4.81 rows=5) (actual time=0.370..0.372 rows=5 loops=1)
                                                            -> Temporary table with deduplication  (cost=2.25..2.25 rows=5) (actual time=0.368..0.368 rows=5 loops=1)
                                                                -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=1.75 rows=5) (actual time=0.245..0.249 rows=
5 loops=1)
                                        -> Single-row index lookup on s using PRIMARY (id=r.song_id)  (cost=0.27 rows=1) (actual time=0.025..0.025 rows=1 loops=3)
                        -> Hash
                            -> Table scan on a  (cost=9988.50 rows=94665) (actual time=0.073..61.248 rows=89741 loops=1)
                    -> Single-row index lookup on <subquery5> using <auto_distinct_key> (id=a.id)  (actual time=0.001..0.001 rows=0 loops=2460)
                        -> Materialize with deduplication  (cost=1.24..1.24 rows=2) (actual time=0.127..0.127 rows=5 loops=1)
                            -> Filter: (song.id is not null)  (cost=1.08 rows=2) (actual time=0.036..0.097 rows=5 loops=1)
                                -> Nested loop inner join  (cost=1.08 rows=2) (actual time=0.035..0.095 rows=5 loops=1)
                                    -> Index lookup on song_history using user_id (user_id='01he4kaj37dam35h94genfkagz')  (cost=0.54 rows=2) (actual time=0.023..0.030 rows=5 loops=1)
                                    -> Single-row covering index lookup on song using PRIMARY (id=song_history.song_id)  (cost=0.31 rows=1) (actual time=0.012..0.013 rows=1 loops=5)
 |
```

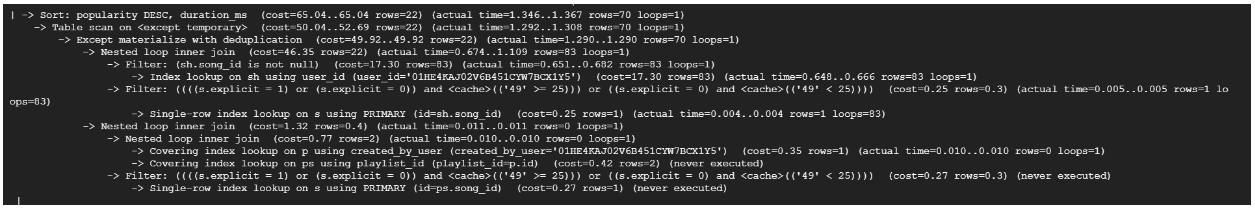
Here we observed, no change in the query cost or execution speed. This might be due the song_name field is used for ordering after the entire result set is generated and hence we can conclude that there was no benefit of adding this index.



```
-------------------------------+
| -> Sort: popularity DESC, duration_ms  (cost=7.51..7.51 rows=1) (actual time=0.138..0.138 rows=1 loops=1)
    -> Table scan on <except temporary>  (cost=7.41..7.41 rows=1) (actual time=0.126..0.126 rows=1 loops=1)
        -> Except materialize with deduplication  (cost=4.91..4.91 rows=1) (actual time=0.124..0.124 rows=1 loops=1)
            -> Nested loop inner join  (cost=3.50 rows=1) (actual time=0.084..0.086 rows=1 loops=1)
                -> Filter: (sh.song_id is not null)  (cost=1.75 rows=5) (actual time=0.048..0.051 rows=5 loops=1)
                    -> Index lookup on sh using user_id (user_id='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=1.75 rows=5) (actual time=0.047..0.049 rows=5 loops=1)
                -> Filter: (((s.explicit = 1) and <cache>(('49' >= 18))) or ((s.explicit = 0) and <cache>(('49' < 18))))  (cost=0.25 rows=0.2) (actual time=0.007..0.007 rows=0 loops=5)
                    -> Single-row index lookup on s using PRIMARY (id=sh.song_id)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=5)
            -> Nested loop inner join  (cost=1.32 rows=0.3) (actual time=0.026..0.026 rows=0 loops=1)
                -> Nested loop inner join  (cost=0.77 rows=2) (actual time=0.025..0.025 rows=0 loops=1)
                    -> Covering index lookup on p using created_by_user (created_by_user='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=0.35 rows=1) (actual time=0.025..0.025 rows=0 loops=1)
                    -> Covering index lookup on ps using playlist_id (playlist_id=p.id)  (cost=0.42 rows=2) (never executed)
                -> Filter: (((s.explicit = 1) and <cache>(('49' >= 18))) or ((s.explicit = 0) and <cache>(('49' < 18))))  (cost=0.26 rows=0.2) (never executed)
                    -> Single-row index lookup on s using PRIMARY (id=ps.song_id)  (cost=0.26 rows=1) (never executed)
 |
```

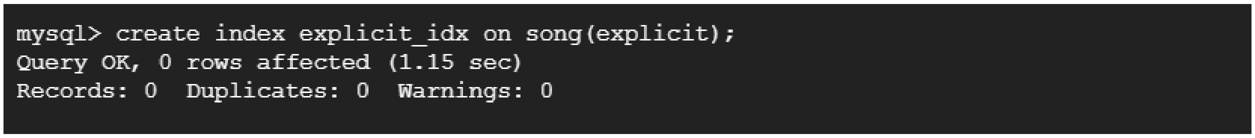
Query 2:

EXPLAIN ANALYSE on regular query

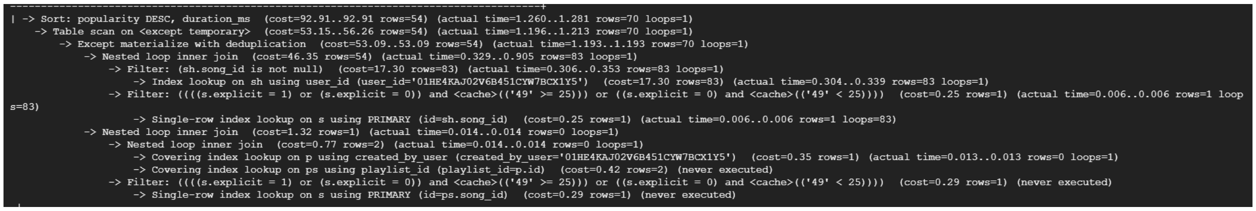


```
| -> Sort: popularity DESC, duration_ms  (cost=65.04..65.04 rows=22) (actual time=1.346..1.367 rows=70 loops=1)
    -> Table scan on <except temporary>  (cost=50.04..52.69 rows=22) (actual time=1.292..1.308 rows=70 loops=1)
        -> Except materialize with deduplication  (cost=49.92..49.92 rows=22) (actual time=1.290..1.290 rows=70 loops=1)
            -> Nested loop inner join  (cost=46.35 rows=22) (actual time=0.674..1.109 rows=83 loops=1)
                -> Filter: (sh.song_id is not null)  (cost=17.30 rows=83) (actual time=0.651..0.682 rows=83 loops=1)
                    -> Index lookup on sh using user_id (user_id='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=17.30 rows=83) (actual time=0.648..0.666 rows=83 loops=1)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.25 rows=0.3) (actual time=0.005..0.005 rows=1 lo
ops=83)
                    -> Single-row index lookup on s using PRIMARY (id=sh.song_id)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=83)
            -> Nested loop inner join  (cost=1.32 rows=0.4) (actual time=0.011..0.011 rows=0 loops=1)
                -> Nested loop inner join  (cost=0.77 rows=2) (actual time=0.010..0.010 rows=0 loops=1)
                    -> Covering index lookup on p using created_by_user (created_by_user='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=0.35 rows=1) (actual time=0.010..0.010 rows=0 loops=1)
                    -> Covering index lookup on ps using playlist_id (playlist_id=p.id)  (cost=0.42 rows=2) (never executed)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.27 rows=0.3) (never executed)
                    -> Single-row index lookup on s using PRIMARY (id=ps.song_id)  (cost=0.27 rows=1) (never executed)
 |
```

Adding index on explicit column in song (Deleting index after **EXPLAIN ANALYSE**)



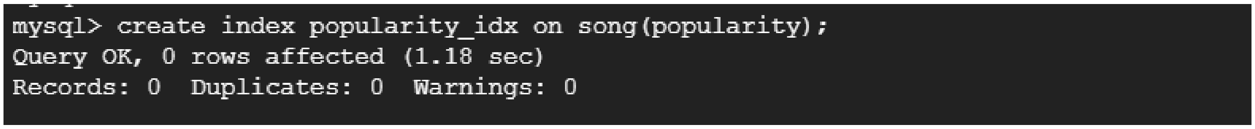
```
mysql> create index explicit_idx on song(explicit);
Query OK, 0 rows affected (1.15 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



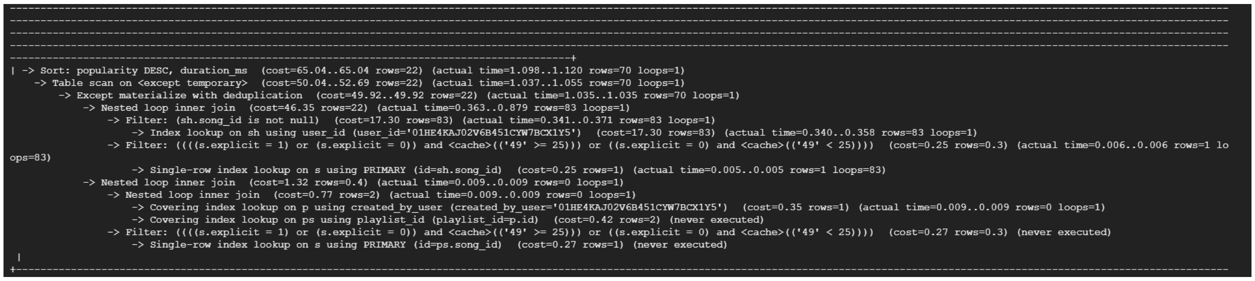
```
----------------------------------------------------------------------------------+
| -> Sort: popularity DESC, duration_ms  (cost=92.91..92.91 rows=54) (actual time=1.260..1.281 rows=70 loops=1)
    -> Table scan on <except temporary>  (cost=53.15..56.26 rows=54) (actual time=1.196..1.213 rows=70 loops=1)
        -> Except materialize with deduplication  (cost=53.09..53.09 rows=54) (actual time=1.193..1.193 rows=70 loops=1)
            -> Nested loop inner join  (cost=46.35 rows=54) (actual time=0.329..0.905 rows=83 loops=1)
                -> Filter: (sh.song_id is not null)  (cost=17.30 rows=83) (actual time=0.306..0.353 rows=83 loops=1)
                    -> Index lookup on sh using user_id (user_id='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=17.30 rows=83) (actual time=0.304..0.339 rows=83 loops=1)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loop
s=83)
                    -> Single-row index lookup on s using PRIMARY (id=sh.song_id)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=83)
            -> Nested loop inner join  (cost=1.32 rows=1) (actual time=0.014..0.014 rows=0 loops=1)
                -> Nested loop inner join  (cost=0.77 rows=2) (actual time=0.014..0.014 rows=0 loops=1)
                    -> Covering index lookup on p using created_by_user (created_by_user='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=0.35 rows=1) (actual time=0.013..0.013 rows=0 loops=1)
                    -> Covering index lookup on ps using playlist_id (playlist_id=p.id)  (cost=0.42 rows=2) (never executed)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.29 rows=1) (never executed)
                    -> Single-row index lookup on s using PRIMARY (id=ps.song_id)  (cost=0.29 rows=1) (never executed)
```

We see that after adding an index on the explicit column, the query execution cost increased rather than reducing. After doing some research we found out that adding an index will not always reduce execution time. Here since we are considering both cases of explicit being 0 and 1 we end up reading almost all records. Since index reads are slower than full table-scans since reads in index are random and cannot take advantage of the read ahead mechanism. We concluded that it is wise to create an index on an attribute of a where clause when we read only a small percentage of records filtered by where.

Index created on popularity (Deleting index after **EXPLAIN ANALYSE**)

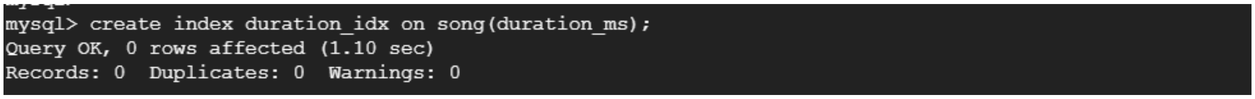


```
mysql> create index popularity_idx on song(popularity);
Query OK, 0 rows affected (1.18 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

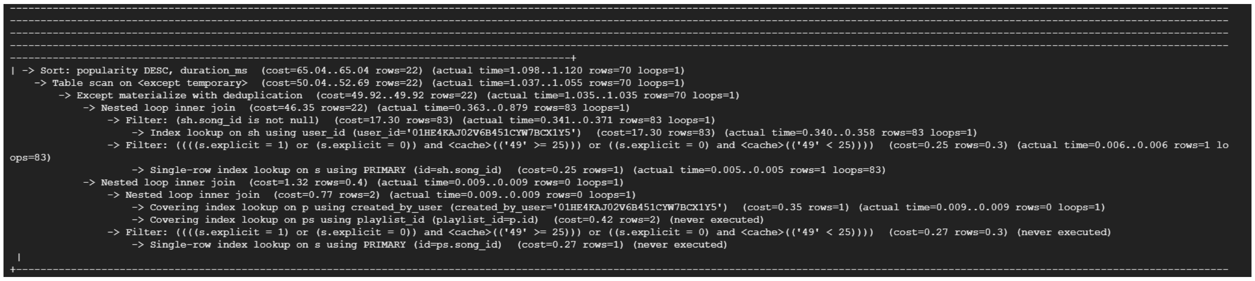


```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
| -> Sort: popularity DESC, duration_ms  (cost=65.04..65.04 rows=22) (actual time=1.098..1.120 rows=70 loops=1)
    -> Table scan on <except temporary>  (cost=50.04..52.69 rows=22) (actual time=1.037..1.055 rows=70 loops=1)
        -> Except materialize with deduplication  (cost=49.92..49.92 rows=22) (actual time=1.035..1.035 rows=70 loops=1)
            -> Nested loop inner join  (cost=46.35 rows=22) (actual time=0.363..0.879 rows=83 loops=1)
                -> Filter: (sh.song_id is not null)  (cost=17.30 rows=83) (actual time=0.341..0.371 rows=83 loops=1)
                    -> Index lookup on sh using user_id (user_id='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=17.30 rows=83) (actual time=0.340..0.358 rows=83 loops=1)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.25 rows=0.3) (actual time=0.006..0.006 rows=1 lo
ops=83)
                    -> Single-row index lookup on s using PRIMARY (id=sh.song_id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=83)
            -> Nested loop inner join  (cost=1.32 rows=0.4) (actual time=0.009..0.009 rows=0 loops=1)
                -> Nested loop inner join  (cost=0.77 rows=2) (actual time=0.009..0.009 rows=0 loops=1)
                    -> Covering index lookup on p using created_by_user (created_by_user='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=0.35 rows=1) (actual time=0.009..0.009 rows=0 loops=1)
                    -> Covering index lookup on ps using playlist_id (playlist_id=p.id)  (cost=0.42 rows=2) (never executed)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.27 rows=0.3) (never executed)
                    -> Single-row index lookup on s using PRIMARY (id=ps.song_id)  (cost=0.27 rows=1) (never executed)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Index created on duration_ms (Deleting index after **EXPLAIN ANALYSE**)



```
mysql> create index duration_idx on song(duration_ms);
Query OK, 0 rows affected (1.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------+
| -> Sort: popularity DESC, duration_ms  (cost=65.04..65.04 rows=22) (actual time=1.098..1.120 rows=70 loops=1)
    -> Table scan on <except temporary>  (cost=50.04..52.69 rows=22) (actual time=1.037..1.055 rows=70 loops=1)
        -> Except materialize with deduplication  (cost=49.92..49.92 rows=22) (actual time=1.035..1.035 rows=70 loops=1)
            -> Nested loop inner join  (cost=46.35 rows=22) (actual time=0.363..0.879 rows=83 loops=1)
                -> Filter: (sh.song_id is not null)  (cost=17.30 rows=83) (actual time=0.341..0.371 rows=83 loops=1)
                    -> Index lookup on sh using user_id (user_id='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=17.30 rows=83) (actual time=0.340..0.358 rows=83 loops=1)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.25 rows=0.3) (actual time=0.006..0.006 rows=1 lo
ops=83)
                    -> Single-row index lookup on s using PRIMARY (id=sh.song_id)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=83)
            -> Nested loop inner join  (cost=1.32 rows=0.4) (actual time=0.009..0.009 rows=0 loops=1)
                -> Nested loop inner join  (cost=0.77 rows=2) (actual time=0.009..0.009 rows=0 loops=1)
                    -> Covering index lookup on p using created_by_user (created_by_user='01HE4KAJ02V6B451CYW7BCX1Y5')  (cost=0.35 rows=1) (actual time=0.009..0.009 rows=0 loops=1)
                    -> Covering index lookup on ps using playlist_id (playlist_id=p.id)  (cost=0.42 rows=2) (never executed)
                -> Filter: ((((s.explicit = 1) or (s.explicit = 0)) and <cache>(('49' >= 25))) or ((s.explicit = 0) and <cache>(('49' < 25))))  (cost=0.27 rows=0.3) (never executed)
                    -> Single-row index lookup on s using PRIMARY (id=ps.song_id)  (cost=0.27 rows=1) (never executed)
 |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
```

For index 2 (popularity) and 3 (duration) here, we added the index on popularity and duration columns from the song table. We added this index to reduce the sorting time in this process. After naively adding the index we noticed there was no change in the query cost and plan as shown in the output below. We noticed that our query was ordering records after using the EXCEPT operation. This maybe due to the optimizer is not leveraging the index as the result set is already Materialized after the except operation and indexing cannot be applied.